INTRODUCTION TO NETWORKED GRAPHICS

SIGGRAPH ASIA, 2011

Anthony Steed

Department of Computer Science, University College London, UK

1. Overview

The Internet has change vastly over the last 10 years. Increasingly, the computer graphics applications we use at work and play are supporting real-time interaction over various types of network. This course introduces the techniques that are used to enable real-time 3D graphics applications such as games and simulators to interact over modern networks. It covers the state of the art in combating latency, bandwidth and scalability constraints.

We take a broad view of networked graphics, including the domains network games, virtual reality and networked simulations. We start by demonstrating why networked graphics applications have different requirements on the network compared to “normal” applications such as web browser. We then describe the problems of providing consistency in networked graphics situations. The main barriers to providing total consistency of views are latency and bandwidth so we discuss ways in which these can be accommodated. Finally we give a survey of other important issues, some other uses of networking and some recent disruptive technologies.

2. Audience

Developers, programmers and analysts interested in networked graphics and its application in games and simulation. Researchers from a range of disciplines interested in the latest state of the art and areas for future development in networked graphics. Students interested in learning more about key technologies behind games, virtual reality and simulations.

If you plan to come to this course, some basic experience of Internet technologies such as standard protocols & services will be helpful. There is a very short re-cap in the first session. We will assume some basic knowledge of computer graphics such coordinate systems & types of 3D model used in real-time systems. Some prior experience playing some real-time networked computer graphics simulations (e.g. games) will have exposed you to some of the issues of consistency that we emphasise during the short course.

3. Lecturer

Anthony Steed is a Professor in the Department of Computer Science at University College London. He leads the Virtual Environments and Computer Graphics research group that numbers around 40 staff and doctoral students. His research interests are in collaborative virtual environments, immersive virtual reality, interaction, and human animation. With Manuel Oliveira he wrote the book Networked Graphics: Building Networked Virtual Environments and Networked Games. In the academic year 2006 - 2007 he was on sabbatical to Electronic Arts in Guildford. He is also the director of the United Kingdom's Engineering Doctorate Centre in Virtual Environments, Imaging, and Visualization.

4. Resources

The lecturer is the main author of the book *Networked Graphics: Building Networked Virtual Environments and Networked Games*, from Elsevier.

This course, and other materials can be found on the Networked Graphics site:

http://www.networkedgraphics.org/

in particular this course including supplemental materials is available here:

http://www.networkedgraphics.org/materials/sigasia2011

That page includes some suggested links for further reading, other sites

Elsevier make freely available from their website Chapter 3 of the Networked Graphics book, which contains a more detailed introduction to the Internet technologies:

http://www.networkedgraphics.org/materials/chapter3

Also that site there are many more lecture slides including slides for a day long course and slides to create an 20-30 hour lecture module for undergraduates or masters-level students). There are additional articles on technologies that have evolved since the book was written and case studies that didn’t make it in to the book. There is a blog that tracks interesting new resources for networked graphics. There are several code examples for you to try.

5. Schedule

This is a half-day (3.75 hour) course.

Course Overview (5 mins)

Introduction (40 mins)

- A very short history of networked graphics
- Requirements for networked graphics
- The Internet and TCP/IP stack
- Basic protocol and architecture choices

Requirements and Constraints (15 mins)

- Internet performance
- Why are all types of networked graphics non-standard networking applications?
- Requirements on consistency
- Implications for latency and scalability

Latency (45 mins)

- Synchronising state with latent communications
- Playout delays, local lag
- Extrapolation and dead reckoning

Break (15 mins)

Bandwidth Management & Scalability (45 mins)

- Bandwidth constraints
- Management of awareness
- Interest specification
- Server partitioning
- Peer to peer networking

Application Support & Recent Research (30 mins)

- Security
- Streaming
- Cluster graphics
- Thin clients
- Scalable peer to peer

Conclusion / Q & A / More Demos (30 mins)

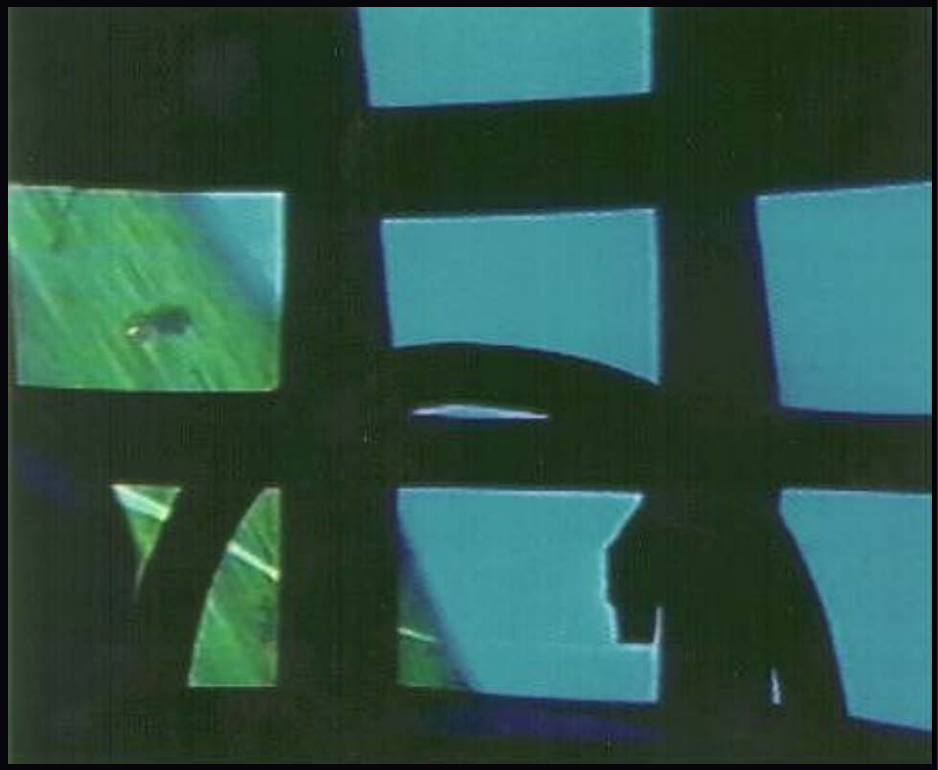
Introduction to Networked Graphics

Part 1 of 5: Introduction

Overview

- Goal:
  - To give an overview of the networked graphics and the Internet
- Topics:
  - A very short history of networked graphics
  - Requirements for networked graphics
  - The Internet and TCP/IP stack
  - Basic protocol and architecture choices

SIMNET

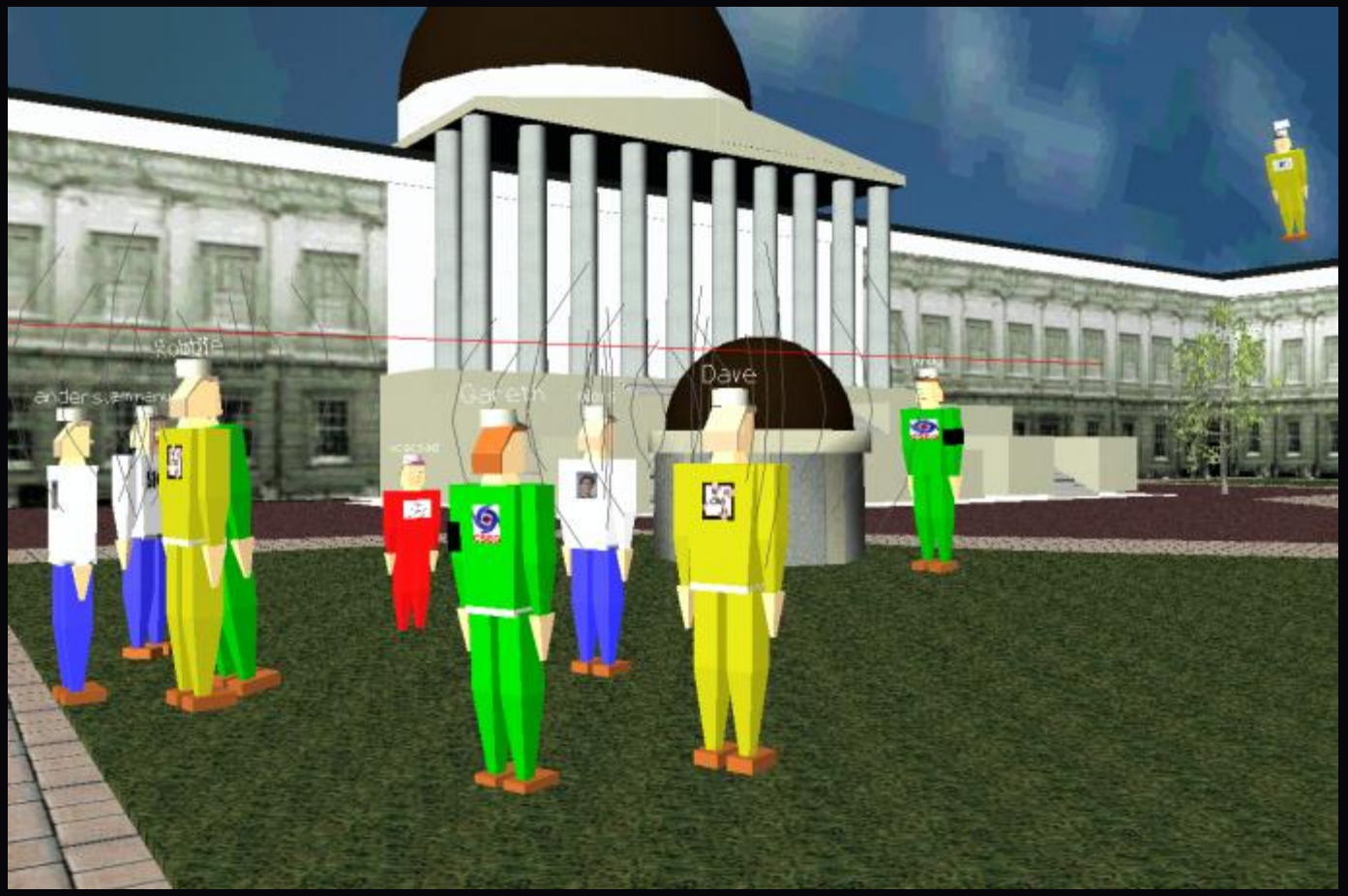


SIMNET aircraft simulation. Left: A view of the simulator control. Right: a view from the cockpit of an aircraft simulator. Images from (Harris, 1994)

DOOM

DOOM™ (iD Software) was the first multi-player first-person shooter to reach wide-spread public attention

DIVE



DIVE system from Swedish Institute of Computer Science. UCL scene in 1999 with spatialised audio amongst 16 participants.

Quake

Quake (id Software) brought true 3D and was the basis for several licensed games. Left: the original Quake game. Right: Counter-Strike (Valve Software), originally a modification of the game Half-Life which was based on Quake engine.

Ultima Online

Pirates demanding tribute (Schultz, A., 2009)

Second Life

Two day conference papers panel in SecondLife

Burnout™ Paradise

Electronic Arts, Burnout™ Paradise

Common Themes

- A shared 3D virtual environment
  - Networked virtual environment (NVE) or networked games (NG)
- Real-time changes
- Collaboration with other users
  - Representation of users in the world (typically as avatars, but also cars/tanks/etc.)
  - Text and occasionally voice communication

Common Themes

- One *client* is usually responsible for generating the view for one user
- A set of clients creates the *illusion* of a shared virtual environment
- "Illusion" because
  - Virtual environments can involve detailed *models*
  - Information about changes in models takes time to travel across communication links

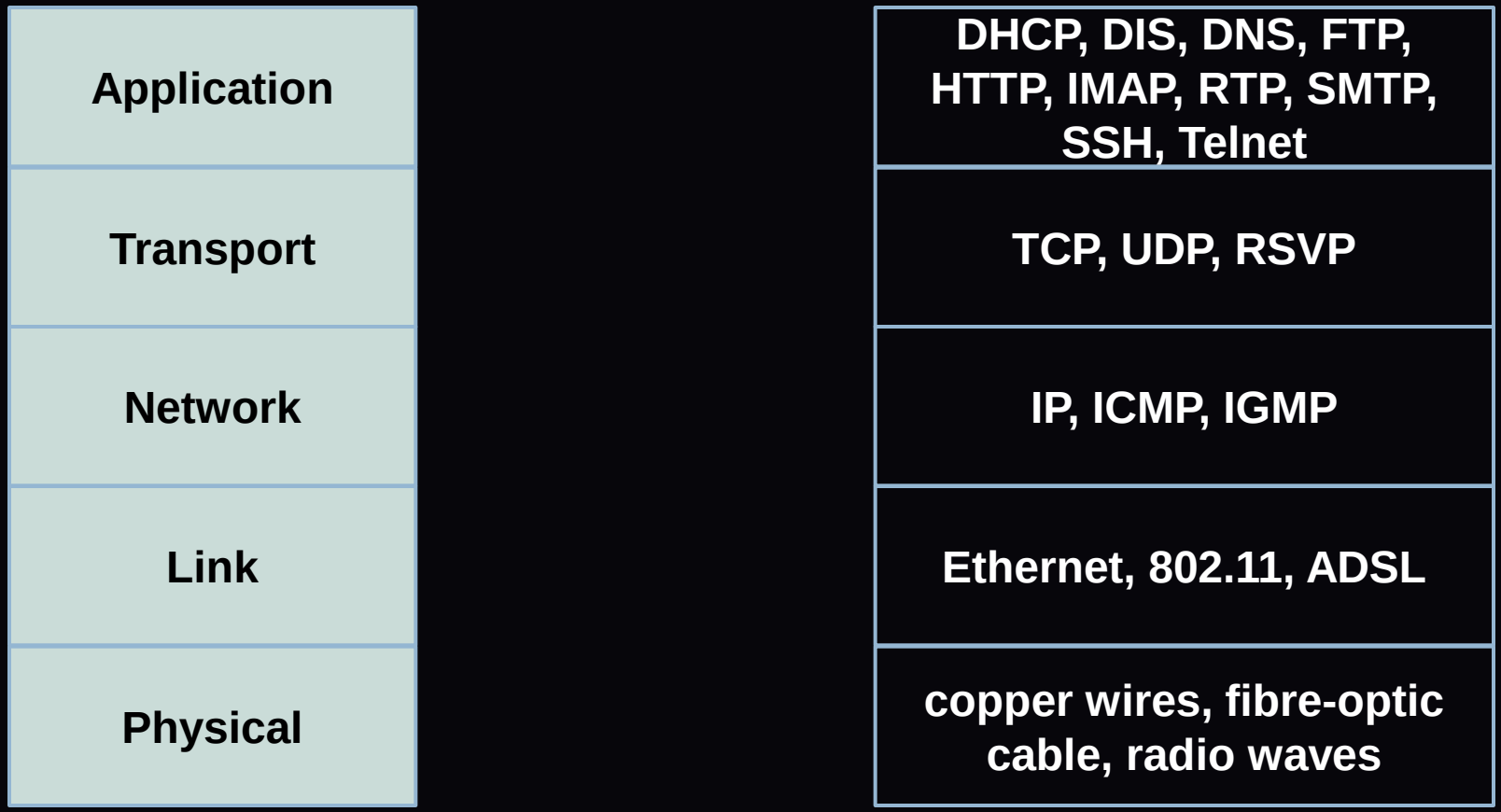
Consistency and Plausibility

- *Local plausibility* is the appearance of consistency of only local actions
- *Shared plausibility* is the appearance of properties being the same as observed by users
  - *Objects that are in the background need not be consistent*
  - *Further: only things that might be the focus of joint attention can be discussed and be different*
- A local implausibility might be an obvious thing to talk about!

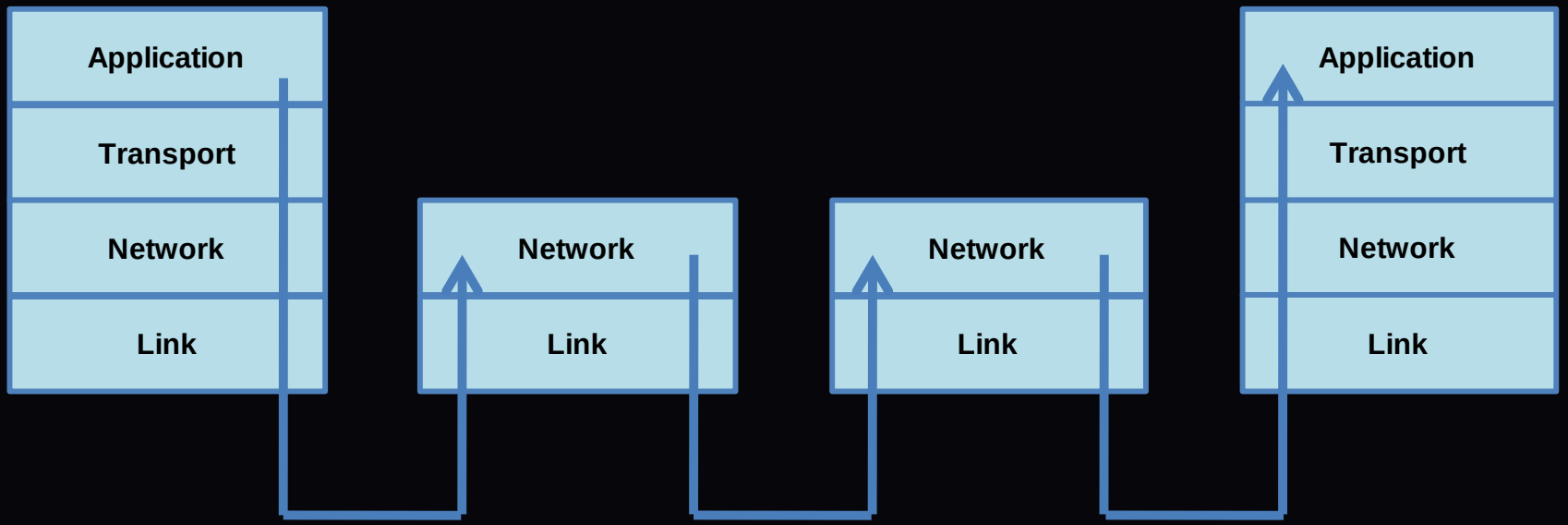
The Internet

- Networks at the heart of *networked* VR
- Many network protocols are out there
  - TCP, UDP, multicast, RTP, etc
  - Choice based on needs
- Properties of the Internet

IP Stack


Application
Transport
Network
Link
Physical
DHCP, DIS, DNS, FTP, HTTP, IMAP, RTP, SMTP, SSH, Telnet
TCP, UDP, RSVP
IP, ICMP, IGMP
Ethernet, 802.11, ADSL
copper wires, fibre-optic cable, radio waves

End to End Principle



- Only the end nodes know about the application. The network only sees IP packets. They don't know about TCP or UDP, or HTTP, etc.

Application Layer Protocols

- Determine what *messages* are sent between applications
  - Messages defined by *syntax* and *semantics*
- Various standards for messages, typically set by RFCs (Requests for Comments) hosted by the IETF (Internet Engineering Task Force)

E.G. HTTP Request

- If you connect to *Host* www.cs.ucl.ac.uk at *Port* 80
- And then issue (type!) in ASCII the following message:

```
GET /staff/A.Steed/ HTTP/1.1
Host: www.cs.ucl.ac.uk
```

- And issues (type) two carriage returns
- You get …

...

HTTP/1.0 200 Document follows

MIME-Version: 1.0

Server: CERN/3.0

Date: Sun, 08 Feb 2009 15:25:18 GMT

Content-Type: text/html

Content-Length: 16150

Last-Modified: Wed, 21 Jan 2009 17:42:00 GMT

<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" lang="en" dir="ltr">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

<meta name="keywords" content="A. Steed, Anthony Steed, Department of Computer Science, University College London, virtual environments, virtual reality, computer graphics" />

Application Protocol Descriptions

- Often ASCII preamble with binary assets inserted at known or marked positions
- Some messages are designed to be carried over a reliable stream and are of unknown length (likely to be over TCP)
- Some messages are small and it is not important if they get lost (likely to be over UDP)

Common Application Protocols

SIGGRAPHASIA2011 HONG KONG

Service Full Name	Short Name	Port	Transport
File Transfer Protocol	ftp	21	tcp
Simple Mail Transfer	smtp	25	tcp
Domain Name System	dns	53	udp
Finger	finger	79	tcp
HyperText Transfer Protocol	http	80	tcp
Post Office Protocol (Version 3)	pop3	110	tcp
Internet Message Access Protocol	imap	143	tcp
Hypertext Transfer Protocol Secure	https	443	tcp
File Transfer Protocol Secure	ftps	990	tcp
Distributed Interactive Simulation	dis	3000	udp
BZFlag Game Server	bzflag	5154	tcp
Quake Game Server	quake	26000	udp

Domain Name Service (DNS)

- Maps fully qualified domain names (narok.cs.ucl.ac.uk) to their IP addresses (128.16.5.123)
- Is a network service, thus takes time
- Time is variable because it's a hierarchical search
- Local DNS caches query responses for a time (e.g. 24 hours)
- Otherwise needs to query a canonical domain

Transport Layer Protocols

- User Datagram Protocol (UDP)
  - Send a message (datagram) and forget about it
  - No guaranteed delivery
  - No guaranteed ordering
- Transmission Control Protocol (TCP)
  - Guaranteed, in-order stream of data from one host to another

UDP

- All hosts on the Internet have an IP address
- How does the network know which program wants it?
- You additionally need (for UDP and TCP) a port number
  - These are 16 bits numbers, so must lie in the range 0-65535
  - Some are reserved, see later
- Processes listen for incoming UDP packets
- Need to check the packet for consistency

TCP

- In comparison to UDP, TCP offers:
  - A connection-oriented services with bi-directional (full-duplex) communication
  - Reliable transmission of messages
  - Congestion avoidance, using variable rate transmission
  - In order, and non-duplicate delivery of information
- Applications add messages to an outgoing buffer
- The buffer is streamed in packets to the receiver
- The receiver reconstructs the buffer and extract packets

TCP is Bi-Directional

- Even if, logically, data only flows one way, in order to ensure reliability, we need return data which tells us which data has been successfully received (ACK)
- The sender must maintain the buffered data until it receives an ACK
- ACKs can waste space if the traffic is mostly one way!

TCP Fairness

- How does TCP decide when to send packets
  - With UDP you call “send”!
- It sends packets within increasing frequency but when they start going missing, it halves its rate
- There are LOTS of variants of TCP
- Protocols are often tested to see if there are TCP-fair, i.e. if N streams share a network link they get 1/N of the bandwidth
- UDP protocols are often NOT TCP-fair, you need to add that functionality yourself

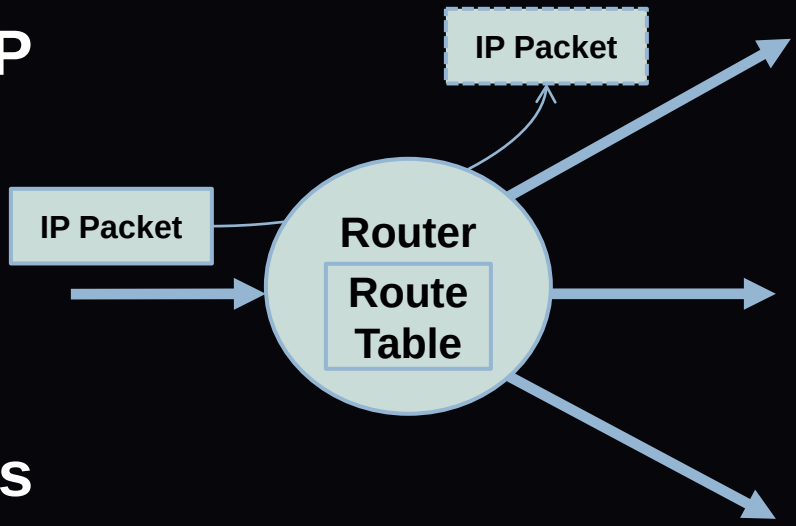
Observations

- If there is lots of data to send TCP can fill up IP packets, UDP might waste network capacity
- There are potentially lots of ACK packets in TCP
- TCP is slow to start, UDP is rapid start
- UDP protocols need to play fair when there is congestion

- Specifically for NVEs & NGs, TCP repair is probably redundant a lot of the time because more up to date data has already been generated by the simulation

Network Layer

- The Internet is a collection of machines that understand IP packets
- A network routes packets from one host to another through routers
- In IPv4 addresses are 32 bits in the form 128.16.13.118
- They are running out and IPv6 is ready to be deployed



IP Packets

- Key problem is what happens if links support frames of different size
  - E.G. Ethernet is 1500bytes
- The solution is that IP supports packet fragmentation, where a large packet is broken in to smaller ones: the end point must then reassemble them
- Obviously try to pick a *maximum transmission unit (MTU)* that avoids this

IP Packet Format

Bits	0			15	16	31
0-31	Version	Header Length	Type of Service		Total Length	
32-63	Identification				Flags	Fragment Offset
64-95	Time to Live		Protocol		Header Checksum	
96-127	Source Address					
128-159	Destination Address					
160-191	Options (Optional)					
160+ 192+, 224+, etc.	Data					

Link and Physical Layer

- The one we all have experience with is Ethernet, either wired or wireless
- Our experience is that for a specific Ethernet interface, we either need to:
  - Set IP address manually
  - Get an address automatically using DHCP
- DHCP is actually an application-layer protocol
- In both cases, we are making a mapping between the MAC address of the Ethernet adapter and the IP address

Basic Architectures and Protocols

- A single connection might be established, but how should a set of clients be connected
- Two basic models are possible
  - Peer to Peer
  - Client/Server
- There are various hybrids that use multiple servers

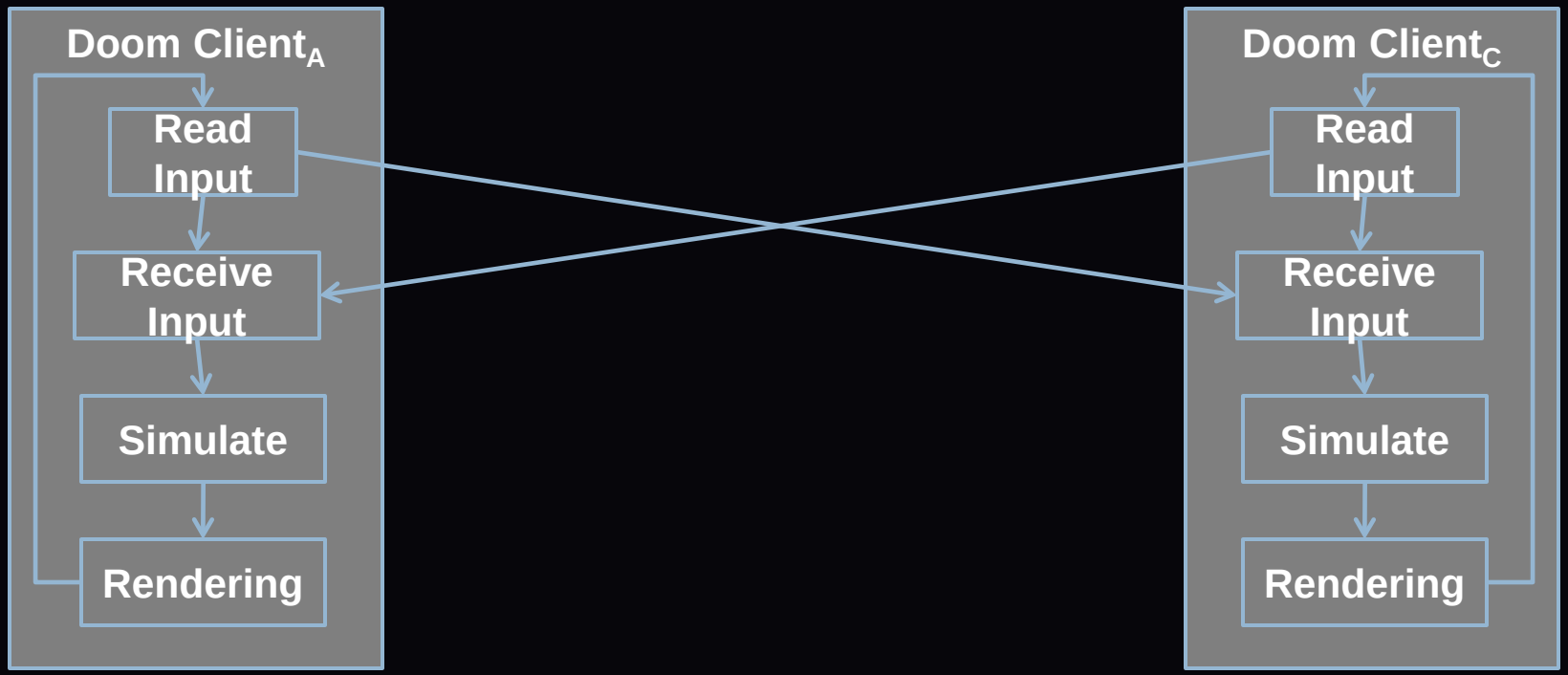
Consider Just Two Machines

- What is the relationship between them?
- Peers?
- Master/slave? Client/server?
- Does one have data the other one does not?

Peer to Peer with Two Clients

- Need to decide separation of responsibilities
  - E.G. Each client simulates one player's actions
- Need to communicate sufficient information to the other that they can get both get the same state
- Assumes that they have the same information other than real-time input
- Can be achieved simply with sending input to each other

For Example DOOM

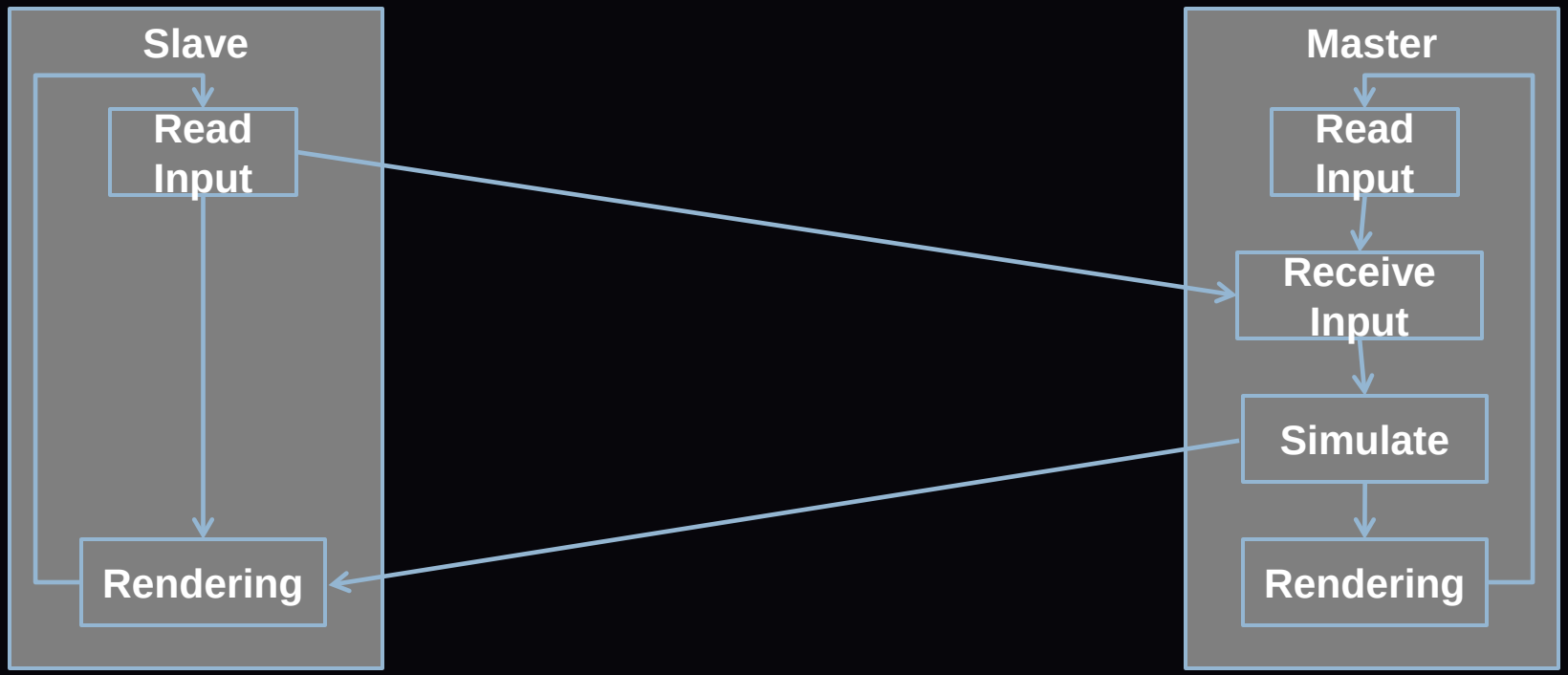


Master Slave with Two Clients

SIGGRAPHASIA2011 HONG KONG

- One process calculates results of input and distributes it to the others
- Necessary if simulation is non-deterministic
- Many examples. E.G. Fable II from Lionhead/ Microsoft Games Studio

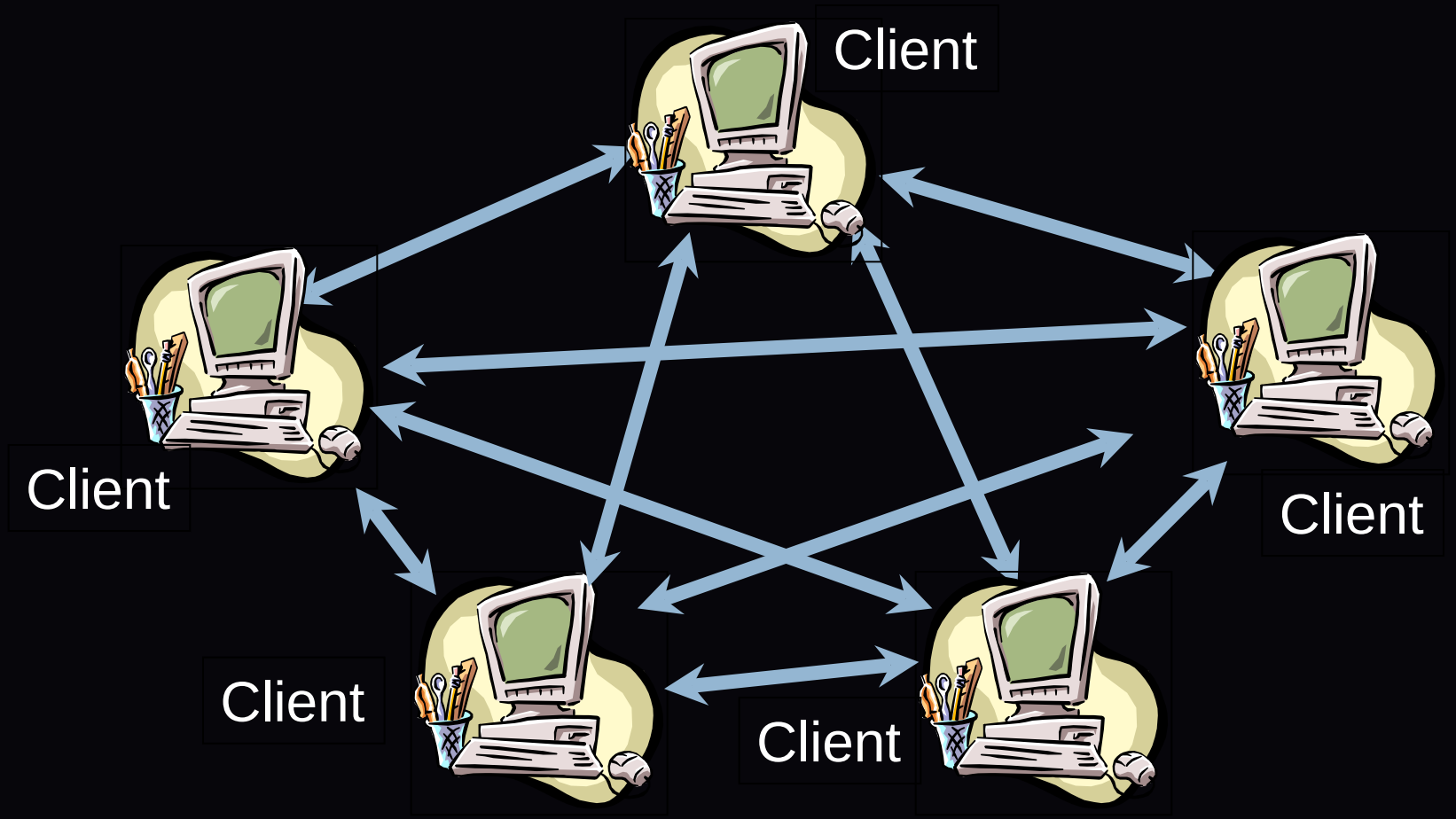
For Example


Slave
Read Input
Rendering
Master
Read Input
Receive Input
Simulate
Rendering

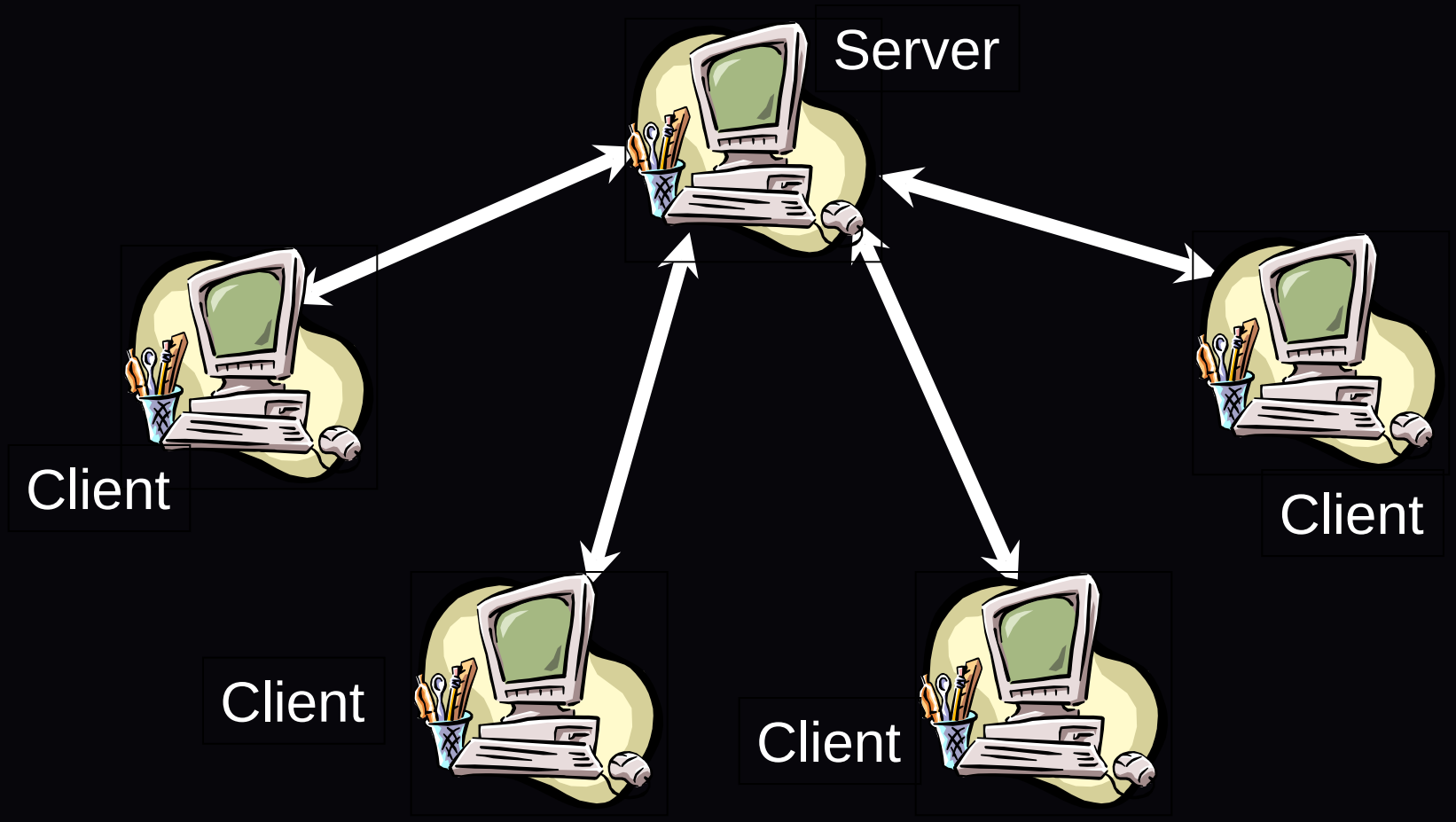
More Clients

- The same issues exist:
  - Who is responsible?
  - Who has the necessary data to evolve the state?
  - Who can be trusted to evolve the state?

Peer to Peer Architecture



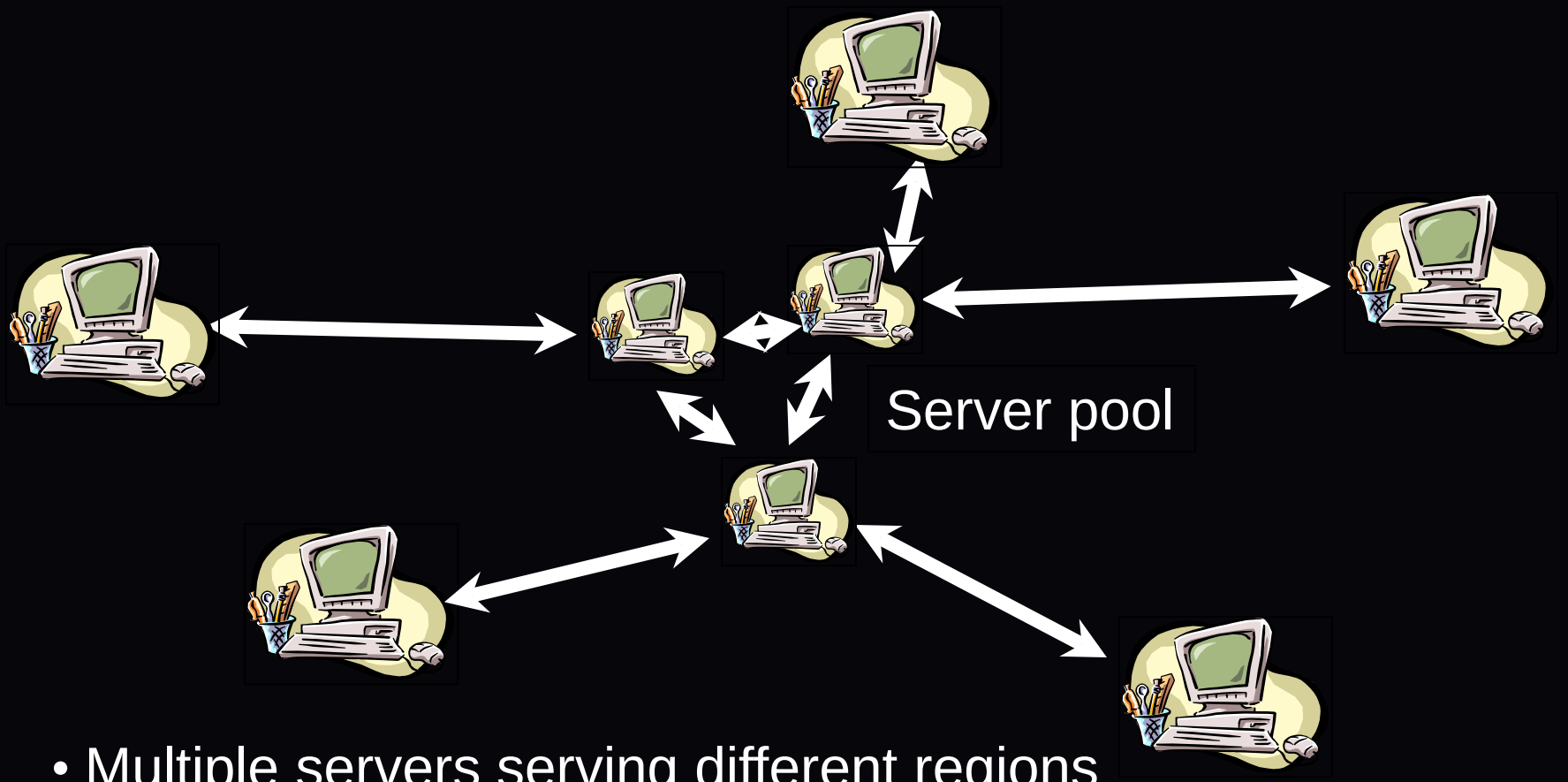
Client-Server Architecture



Implications

- Peer to Peer
  - Data need to be sent multiple times on the network links might vary in bandwidth & latency
  - Clients need to manage multiple connections
- Client Server
  - The Server is a bottleneck
  - Clients manage one connection
  - Server can have privileged data, and can probably be trusted
  - Latency is higher
  - Synchronization is easy

Hybrid Architectures



- Multiple servers serving different regions
- Multiple service types & service layers

Which Protocol to Use?

- If there is an application layer protocol that is appropriate use that!
- UDP
  - Good for fast changing data, and initial start update
  - Good for position information
- TCP
  - Good for reliable data, and bulk data transfer
  - Good for data assets and critical information such as score

Which Protocol to Use?

- Some people implement “reliability-lite” on top of UDP
- Other platforms mix UDP & TCP
  - There are many catches with this
- Many platforms support application layer protocols such as HTTP or FTP for bulk asset transfer

TCP is Bi-Directional

- Even if, logically, data only flows one way, in order to ensure reliability, we need return data which tells us which data has been successfully received (ACK)
- The sender must maintain the buffered data until it receives an ACK
- ACKs can waste space if the traffic is mostly one way!

TCP Fairness

- How does TCP decide when to send packets (with UDP you call “send”)?
- It sends packets within increasing frequency but when they start going missing, it halves its rate
- There are LOTS of variants of TCP
- Protocols are often tested to see if there are TCP-fair, i.e. if N streams share a network link they get 1/N of the bandwidth
- UDP protocols are often NOT TCP-fair, you need to add that functionality yourself

Observations

- If there is lots of data to send TCP can fill up IP packets, UDP might waste network capacity
- There are potentially lots of ACK packets in TCP
- TCP is slow to start, UDP is rapid start
- UDP protocols need to play fair when there is congestion

Summary

- NVEs & NGs have a long history, but it is in the last 10 years that they have really taken off
- The Internet is a *best effort* network where applications need to deal with latency & loss
- There are various architectures that support NVEs & NGs
  - Client server versus peer to peer
  - As much a question of delegation of responsibility as connectivity
  - TCP v. UDP debate

Introduction to Networked Graphics

Part 2 of 5: Requirements and Constraints

Overview

- Goal:
  - To give an overview of the performance of the Internet and how it affects how NVEs can work.

- Topics:
  - Internet performance
  - Why are all types of networked graphics non-standard networking applications?
  - Requirements on consistency
  - Implications for latency and scalability

Internet performance

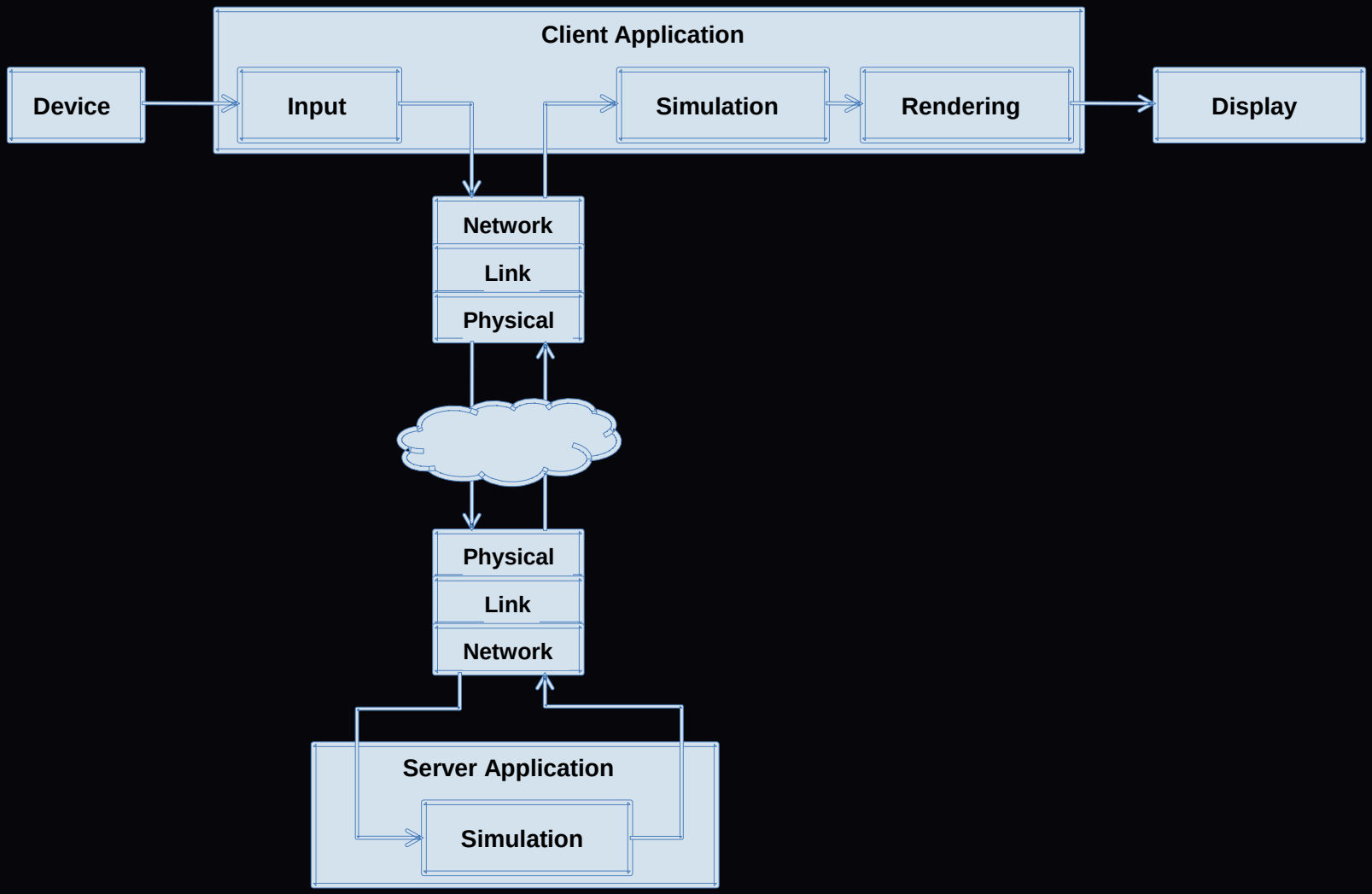
3

- Latency (Round Trip Time)
  - Time to transmit data (speed of light, modems)
- Jitter
  - Routers insert bandwidth
- Bandwidth (Capacity)
  - Broadband for WAN, Ethernet for LAN?
- Loss (Congestion, Reliability)
  - Routers drop packets, links do go down
- Not fully connected
  - Network address translation

Latency

- Sources of latency
  - Speed of copying to link (e.g. modem)
  - Speed of transmission in link (e.g. speed of light)
  - Client scheduling (when packets arrive compared to the commitment to render the effect)
  - Server scheduling (e.g. server updates at a fixed frequency)

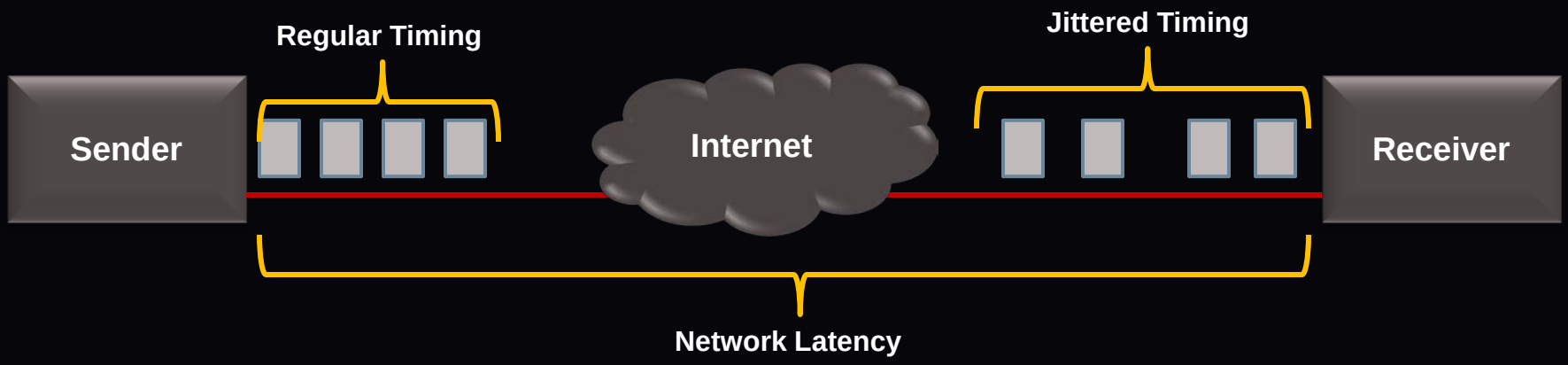
Latency


Client Application
Device
Input
Simulation
Rendering
Display
Network
Link
Physical
Physical
Link
Network
Server Application
Simulation

Jitter

- Jitter is change in latency
- Jitter is caused by the technology of the Internet
  - Wired routers
  - Wireless access
- Two problems:
  - Routers are almost certainly capacity bound and demand on routers changes rapidly
  - Some link layers (notably wireless) are shared medium so transmitters will conflict

Latency and Jitter : Network Perspective



Transmission Delay : time it takes to put a packet on the outgoing link
Propagation Delay : time it takes for the packet to arrive at destination

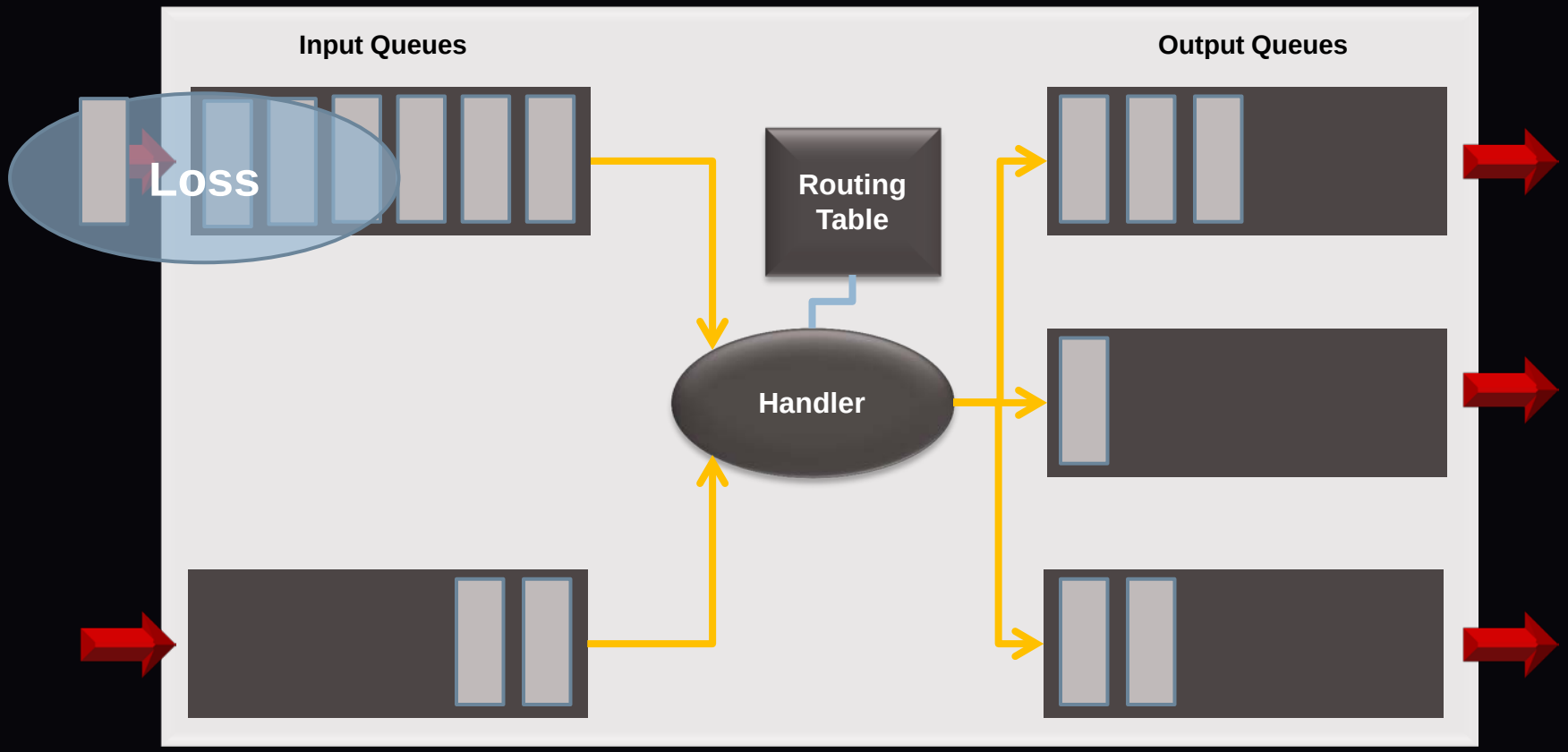
Bandwidth

- Bandwidth is a shared resource
- At local level we shared the wireless or share a home or office router
  - Can be much more outbound or requested inbound traffic that the local network can access
- However probably, the bottleneck is likely to be upstream to our ISP
- ISP have intra-ISP (and “senior” ISP) bottlenecks
- The destination site (BBC, Facebook) might have inbound capacity limits

Loss

- Loss is good
- Loss is the Internet's way of protecting itself from overload
- Principally caused by congestion: a router can't cope with the throughput OR it can't copy all incoming traffic on to the desired outgoing route
- End to end protocols need to detect loss
  - TCP does this for you
- Protocols need to rate limit because not doing so will likely make the situation worse

Loss : Network Perspective



Bandwidth and Latency: Wired

SIGGRAPHASIA2011
HONG KONG

- Much literature in the area is based on 56kbps modems …
- Broadband is now common in homes
  - 500Kbps – 1Gbps
  - Depends on technology (twisted-pair v. optical)
- Offices have always been different
  - 1Gbps Ethernet, switched (not shared) is common
  - Outbound varies enormously

- Latency is good

Bandwidth and Latency: Wireless

SIGGRAPHASIA2011 HONG KONG

- 2G
  - Don’t try, run web or sms-based applications!
- 3G / 4G
  - 3G: ~2.4Mbps
  - 4G: 100Mbps – 1Gbps
- 802.11a-n
  - b: 11 Mbps
  - n: 54 Mbps
- Be skeptical: its shared bandwidth
- Latency is moderate-poor: its shared bandwidth

Bandwidth Availability

Rank	Country	Mbps Q1, 2011
-	Global	2.1
1	South Korea	14.4
2	Hong Kong	9.2
3	Japan	8.1
4	Netherlands	7.5
5	Romania	6.6
6	Czech Rep.	6.5
7	Latvia	6.3
8	Switzerland	6.2
9	Belgium	6.1
10	Ireland	5.6
…		
14	United States	5.3

Average connection speed by country, Q1 2011. Based on Akamai, State of the Internet, 4(2)

Effect of distance on throughput

SIGGRAPHASIA2011 HONG KONG

Distance from Server to User (miles)	Network Latency (ms)	Typical Packet Loss (%)	Throughput (Mbps) :Quality	4GB DVD Download Time
Local: <100	1.6	0.6	44:HDTV	12min
Regional: 500-1,000	16	0.7	4:Almost DVD	2.2hrs
Cross-continent ~3,000	48	1.0	1:Almost TV	8.2hrs
Multi-continent ~6,000	96	1.4	0.4:Poor	20hrs

Based on (Leighton, 2009)

Why NVEs are unique

- NVEs are not “standard” network applications
- Unlike video/audio streaming, or web browsing, in an NVE or NG client, networking is NOT the main activity: rendering probably is
- Some information changes very quickly and smoothly
  - E.G. player positions
- Can incorporate other web-enabled media
  - Audio/video
- Often require bulk download of assets
- NVEs mix different types of requirement

Why NVEs are unique

- Internet is built to move bulk traffic, but not for end to end speed
- You can’t reserve bandwidth (except in certain situations)
- Latency and bandwidth will vary
- Streaming for audio and video will buffer significantly and loss is not important
- For NVEs loss can be critical, but also buffering is usually not appropriate

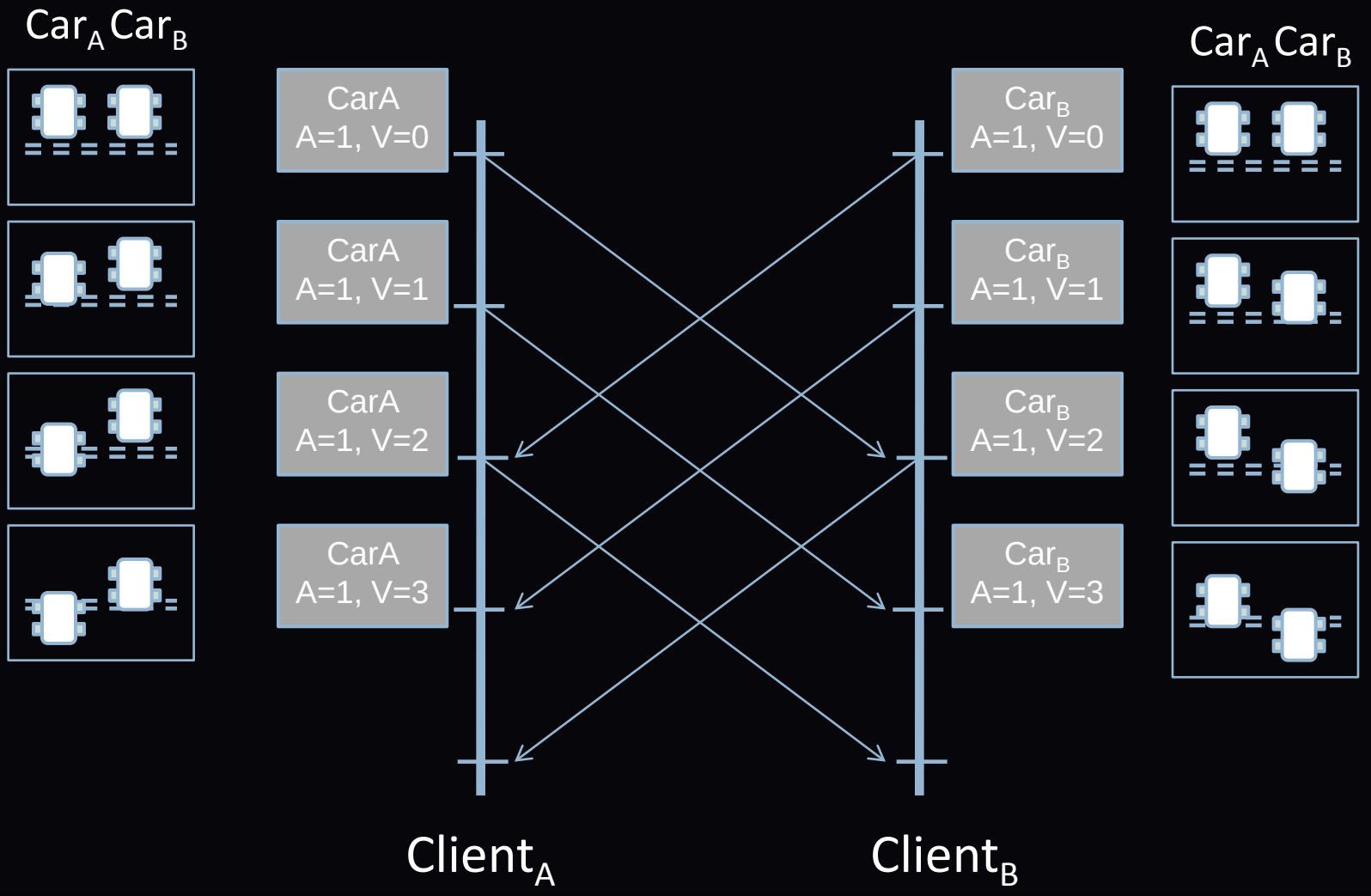
Consistency : System Perspective

- C1 : Local changes replicated at each site
- C2 : Simulation should not diverge over time
- C3 : Casual order of events should be preserved
- C4 : Temporal and motion characteristics of events should be preserved

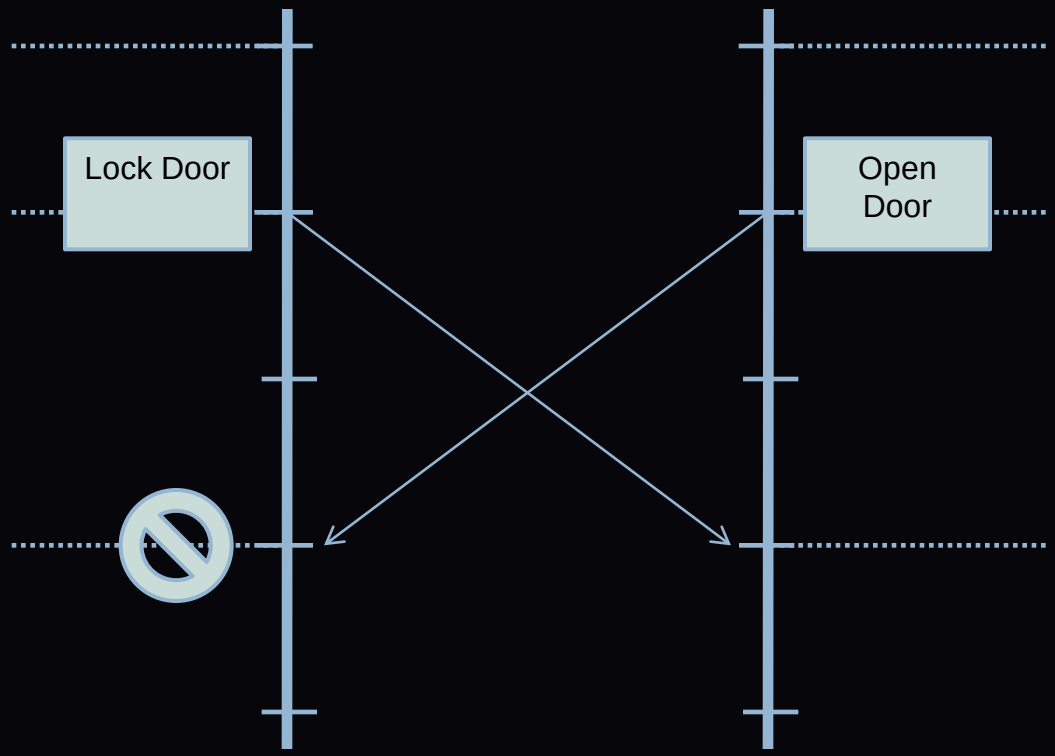
Consistency : User Perspective

- C5 : The joint perception of events should be plausible
- C6 : The outcome of the events should be fair
- C7 : The system should preserve the users' intentions

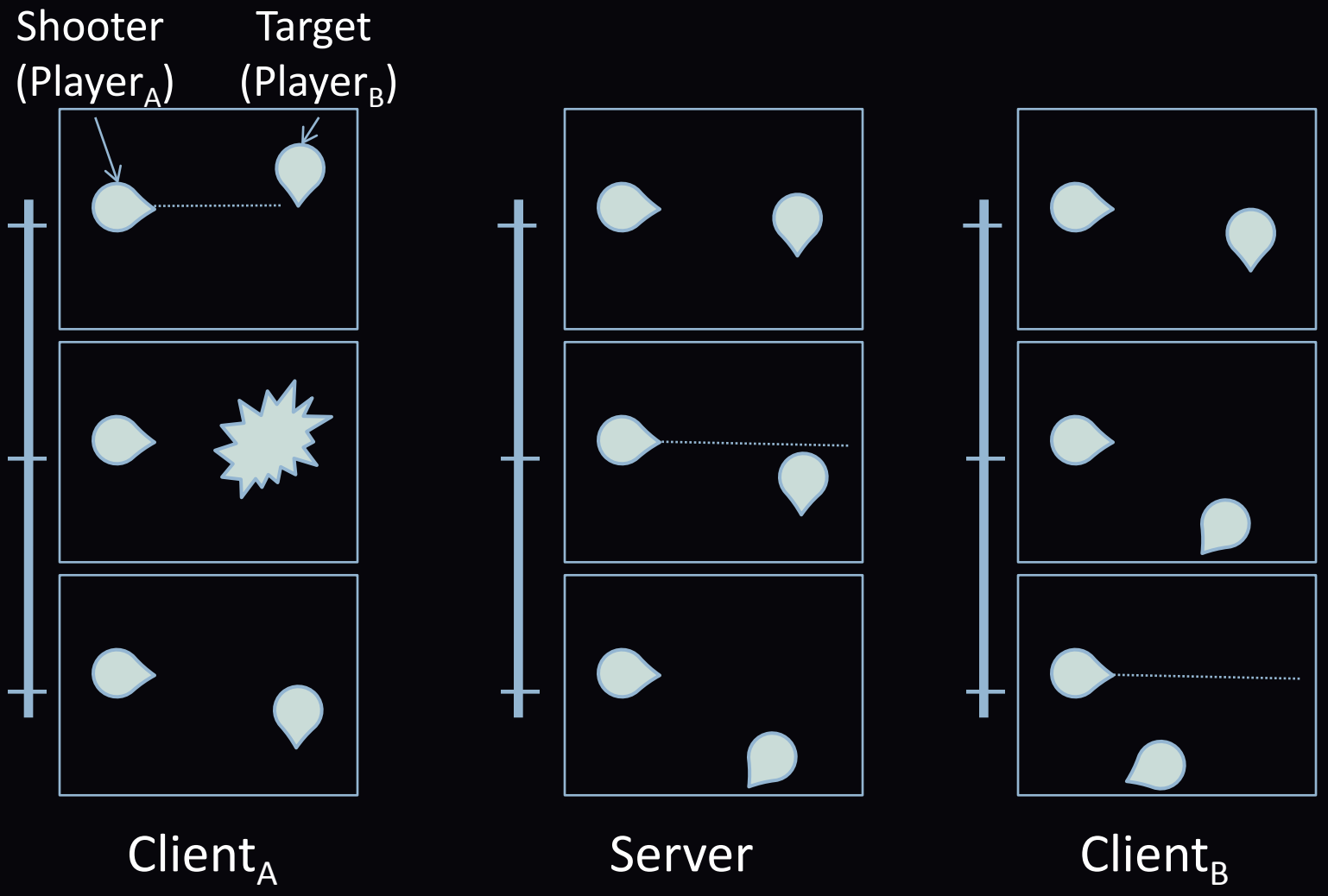
Impact: Timing Activity Onset



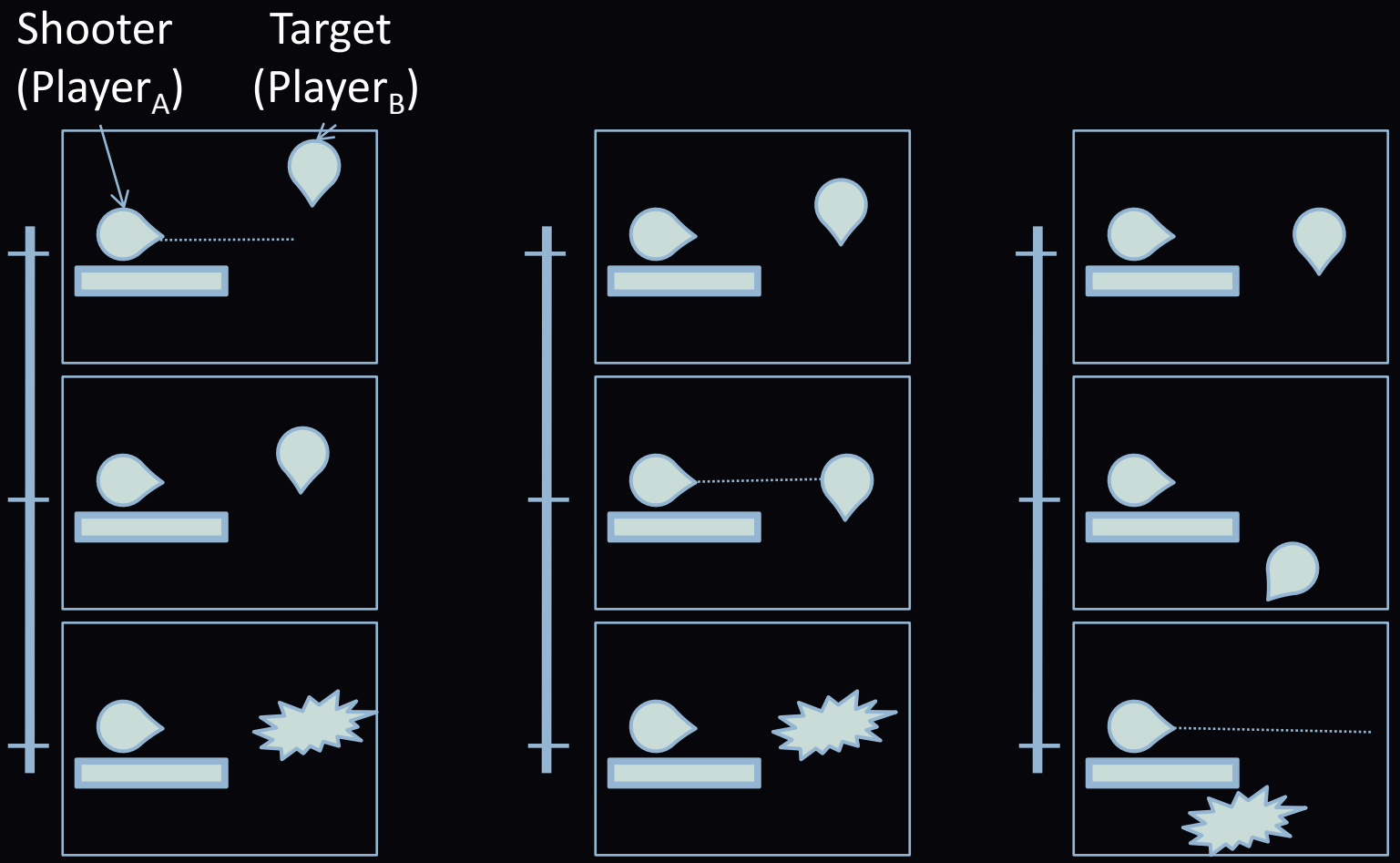
Impact: Inconsistent State Changes



Impact: Fireproof Players



Impact: Shooting Around Corners



Latency Acceptability

Several tasks plotted on the Precision/Deadline axes. Based on Claypool and Claypool (2006).

Bandwidth Requirements

SIGGRAPHASIA2011
HONG KONG

- Obviously depends on activity
  - Downloading models
  - Sending small, game specific commands
  - Rate of command sending (very sensitive to type of game)
- Typically:
  - FPS & real-time send commands at fixed rate (e.g. 20 Hz)
  - RTS and other send commands at issue rate (e.g. up to 5Hz with StarCraft)

Packet Rates

Game	Packet Rate In (pps)	Packet Rate Out (pps)	Packet Size In (bits)	Packet Size Out (bits)
Day of Defeat	421.85	341.92	41.73	162.78
Medal of Honor: Allied Assault	379.67	294.10	50.10	291.71
Unreal Tournament 2003	469.89	123.43	27.92	117.74

Server packet rates and sizes for three FPS games, from Feng et al. (2005)

Packet Rates

Game	Packet Rate In (pps)	Packet Rate Out (pps)	Packet Size In (bytes)	Packet Size Out (bytes)
World of Warcraft	6.39	6.21	220.25	71.12
Guild Wars	3.76	3.83	183.19	57.78
Eve Online	0.84	0.86	261.18	64.41
Star Wars Galaxies	12.26	6.34	156.47	77.25

Client packet rates and sizes for four MMORPG games, from Molnár & Szabó (2008)

Packet Rates

Zone Type	Direction	Standing (kbps)	Walking (kbps)	Teleport (kbps)	Flying (kbps)
Dense & Crowded	S-C	192	703	1164	877
	C-S	15	31	33	31
Dense & Deserted	S-C	141	278	445	821
	C-S	30	46	36	52
Sparse & Deserted	S-C	10	31	448	27
	C-S	13	74	36	73

Bandwidth of Second Life for different region types and different modes of travel. From Kinicki & Claypool (2008)

Network Address Translation

- The biggest hiccup for any peer to peer networking
- Many (most?) computers on the Internet are behind a NAT
- We are behind a NAT
  - 192.168.14.32 is in a *reserved* IP address domain
- Your home network probably runs a NAT
  - You have one address from your ISP
  - You might pay to have this be a static IP address
- NATs complicate

Comments on NATs

- Many types of NAT, port static, symmetric, etc.
- You can bypass NATs with “hole-punching” techniques
- Many game middleware have a function for this BUT
  - Game providers need to provide a rendezvous service
  - Need a packet relay service when it fails
- For a peer to peer game, middleware tries to assess which client has best connectivity
- NATs often are combined with the functionality of firewalls whose role is to protect the LAN from malicious incoming traffic

Summary

- Broadband accessibility is growing
- NVEs and NGs tend to demand a lot from the network
  - Some games have low latency requirements
  - Packet rates vary enormously depending on the game type
- The immediate impact of Internet performance can lead to de-synchronization and player frustration
- The Internet is not symmetrically connected

Introduction to Networked Graphics

Part 3 of 5: Latency

Overview

- Goal:
  - To explain how latency impacts the decisions of how to ensure consistency. Latency implies that clients cannot all act the same way because they don't have consistent information.

- Topics:
  - Synchronising state with latent communications
  - Playout delays, local lag
  - Extrapolation and dead reckoning

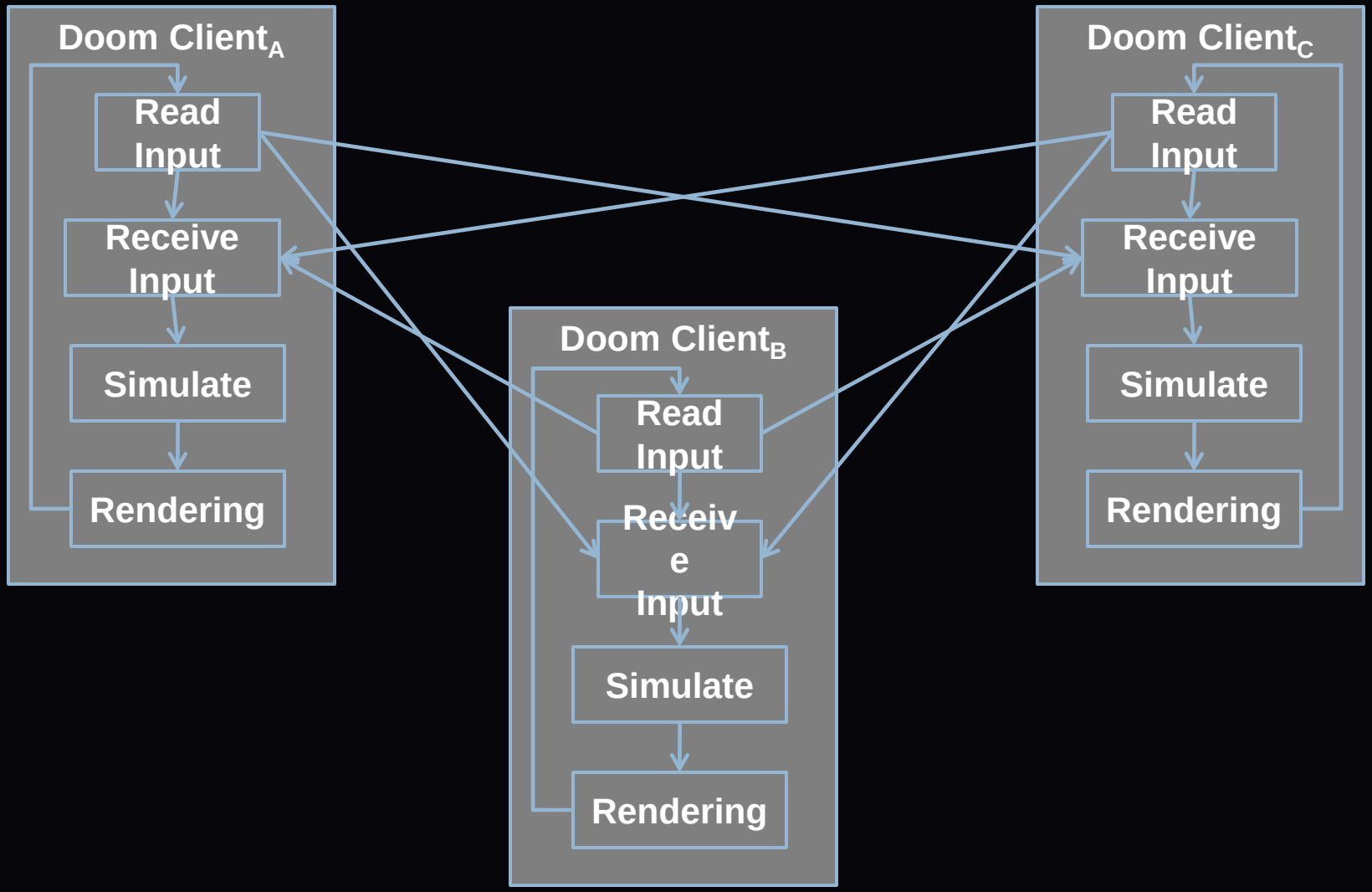
Naïve (But Usable) Algorithms

- Most naïve way to ensure consistency is to allow only one application to evolve state at once
- One application sends its state, the others wait to receive, then one proceeds
- Is a usable protocol for slow simulations, e.g. games
  - Not that slow – moves progress at the inter-client latency
- Potentially useful in situations where clients use very different code, and where clients are “un-predictable”

Lock-Step (1)

- If all clients can deterministically on the input data
- Then a more useful form lock-step for NVEs & NGs is that everyone exchange input, proceed once you have all the information from other clients
- But for many simulations, each step is only determined by user input, so can just communicate input

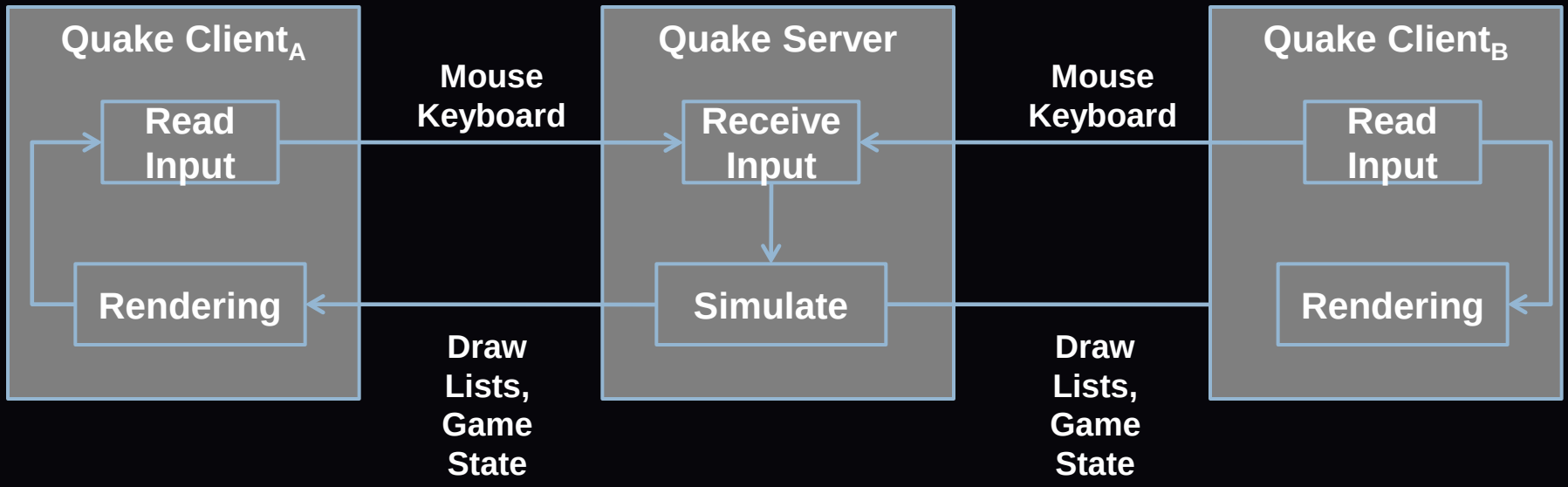
DOOM 1 – iD Software



Lock-Step (2)

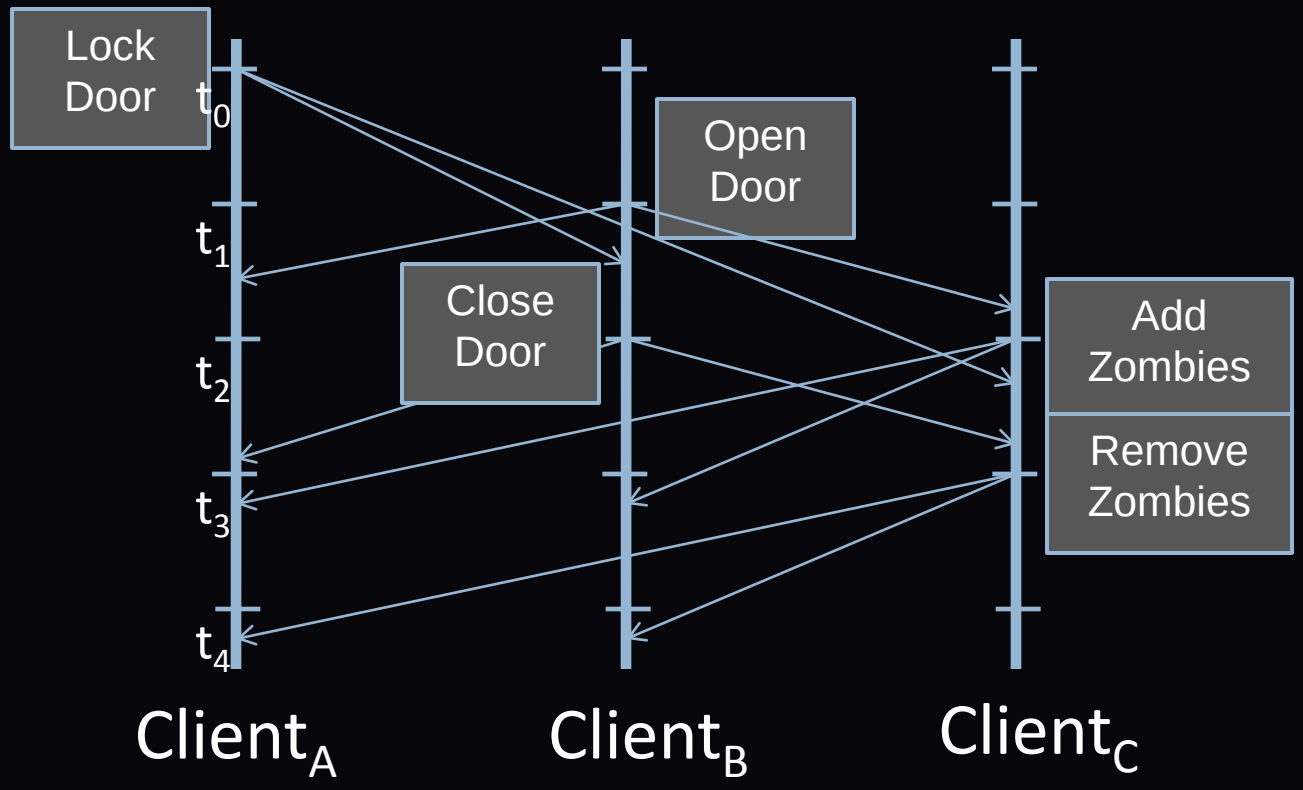
- If the simulation is complex or non-deterministic, use a server to compute the state
- Clients are locked to the update rate of the server
- Note that *own input* is delayed

Quake 1 (Pre-QuakeWorld)



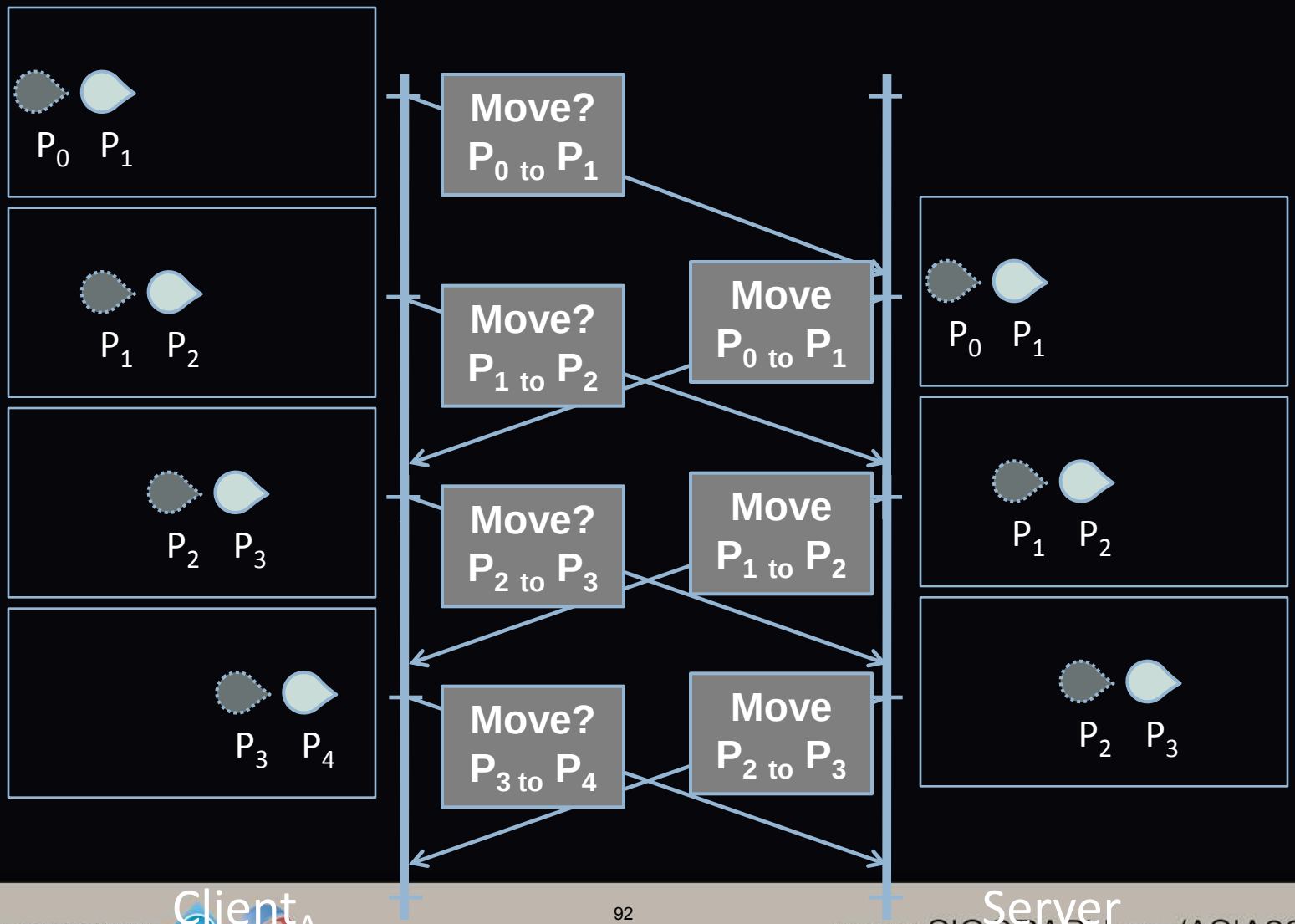
Optimistic Algorithms

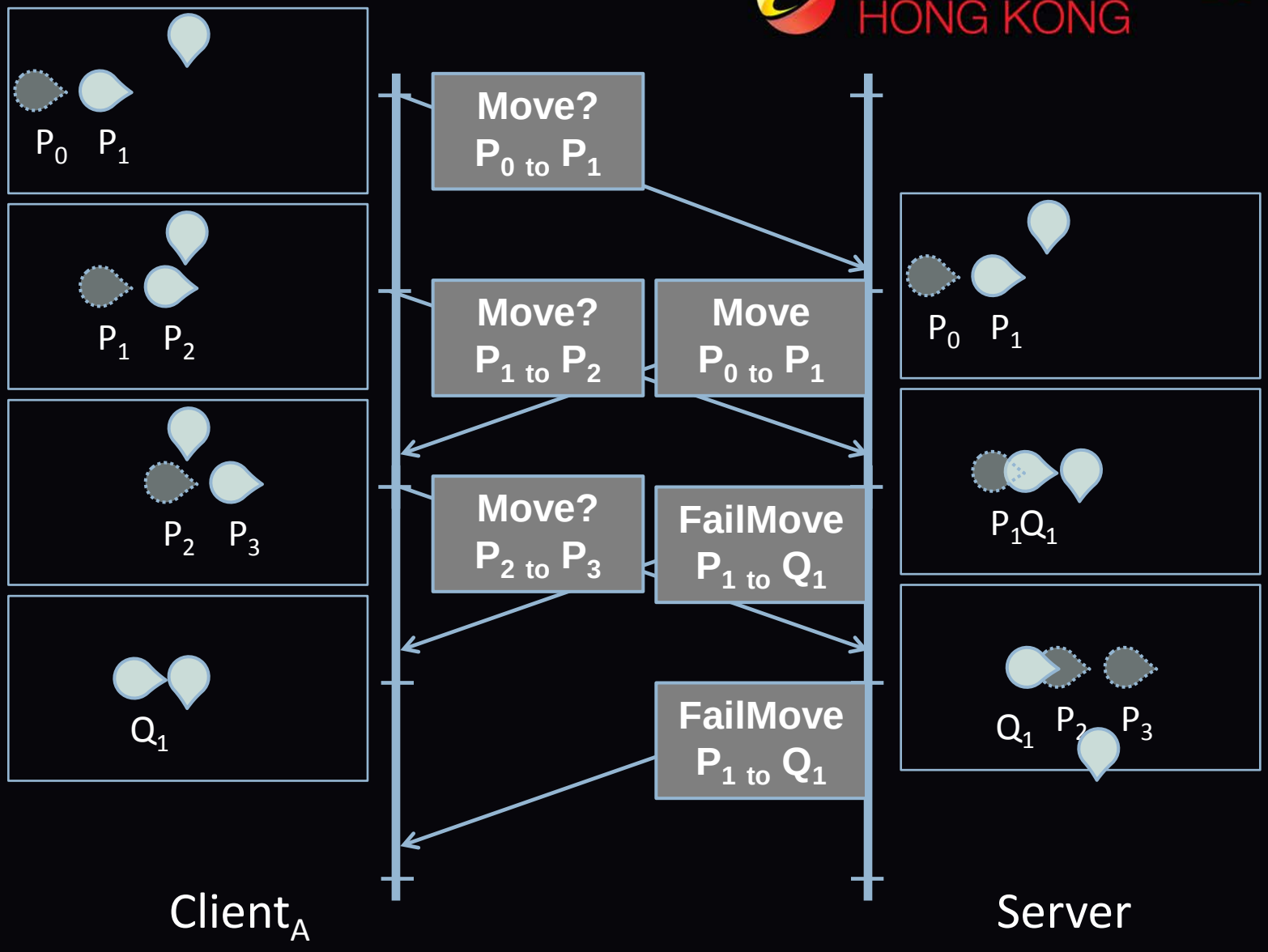
- Conservative simulations tend to be slowed paced
- Optimistic algorithms play out events as soon as possible
- Of course, this means that they can get things wrong:
  - They may receive an event that happened in the past
  - To fix this they rollback by sending UNDO events
  - For many simulations UNDO is easy (just move something)


Lock Door
t_0
t_1
t_2
t_3
t_4
Open Door
Close Door
Add Zombies
Remove Zombies
$Client_A$
$Client_B$
$Client_C$

Client Predict Ahead

- A form of optimism: assume that you can predict what a server (or another peer) is going to do with your simulation
- Very commonly applied in games & simulations for your own player/vehicle movement
- You assume that your control input (e.g. move forward) is going to be accepted by the server
- If it isn't, then you are moved back Note this isn't *forwards in time* but a prediction of the current canonical state (which isn't yet known!)


P_0 P_1
P_1 P_2
P_2 P_3
P_3 P_4
Move? P_0 to P_1
Move? P_1 to P_2
Move? P_2 to P_3
Move? P_3 to P_4
Move P_0 to P_1
Move P_1 to P_2
Move P_2 to P_3
P_0 P_1
P_1 P_2
P_2 P_3
Client A
Server


P0 P1
P1 P2
P2 P3
Q1
Move?
P0 to P1
Move?
P1 to P2
Move
P0 to P1
Move?
P2 to P3
FailMove
P1 to Q1
FailMove
P1 to Q1
P0 P1
P1Q1
Q1 P2 P3
ClientA
Server

Extrapolation Algorithms

- Because we “see” the historic events of remote clients, can we predict further ahead (i.e. in to their future!)
- This is most commonly done for position and velocity, in which case it is known as *dead-reckoning*
- You know the position and velocity at a previous time, so where should it be now?
- Two requirements:
  - Extrapolation algorithm: how to predict?
  - Convergence algorithm: what if you got it wrong?

Dead Reckoning: Extrapolation

- 1st order model

$$P_1 = P_0 + (t_1 - t_0).V_0$$

- 2nd order model

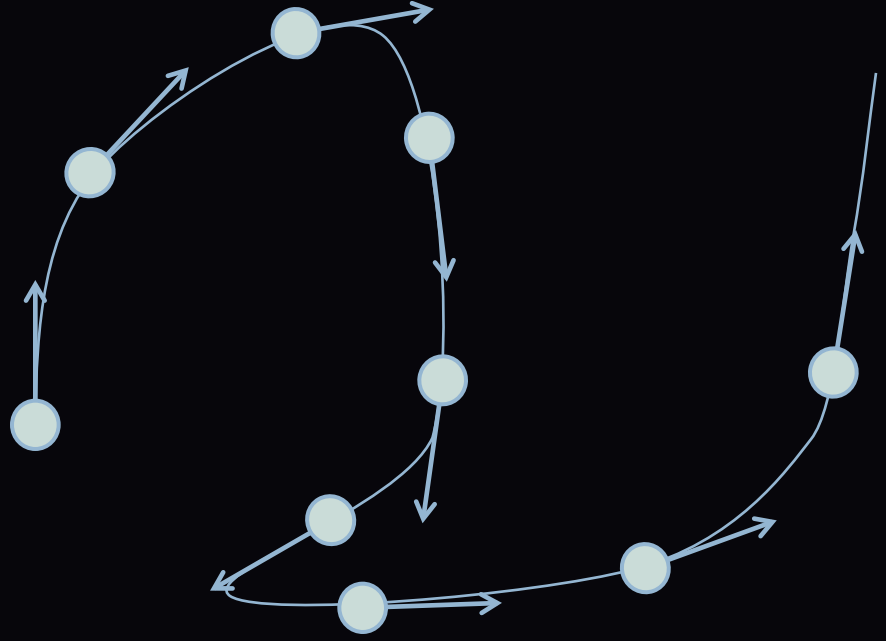
$$V_1 = V_0 + (t_1 - t_0).A_0$$

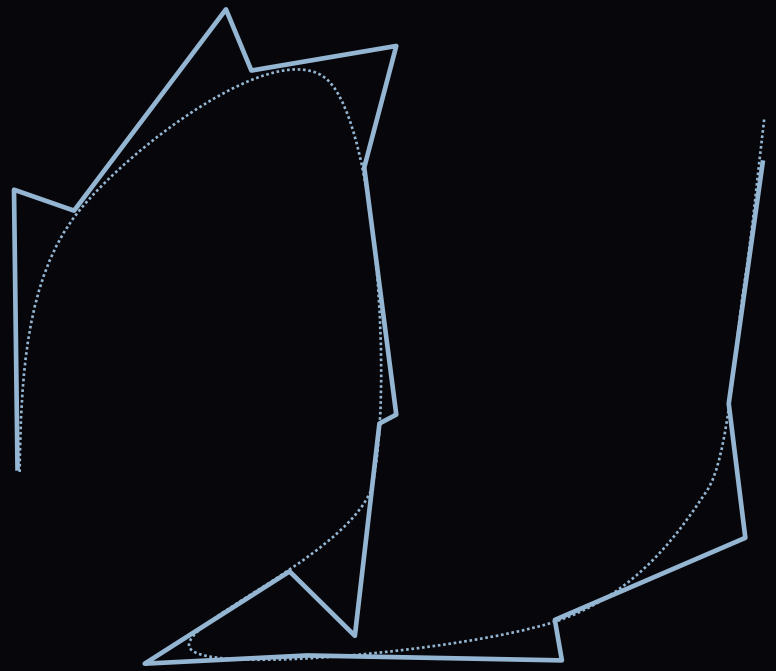
$$P_1 = P_0 + (t_1 - t_0).V_0 + \frac{1}{2}.A_0.(t_1 - t_0)^2$$

When to Send Updates

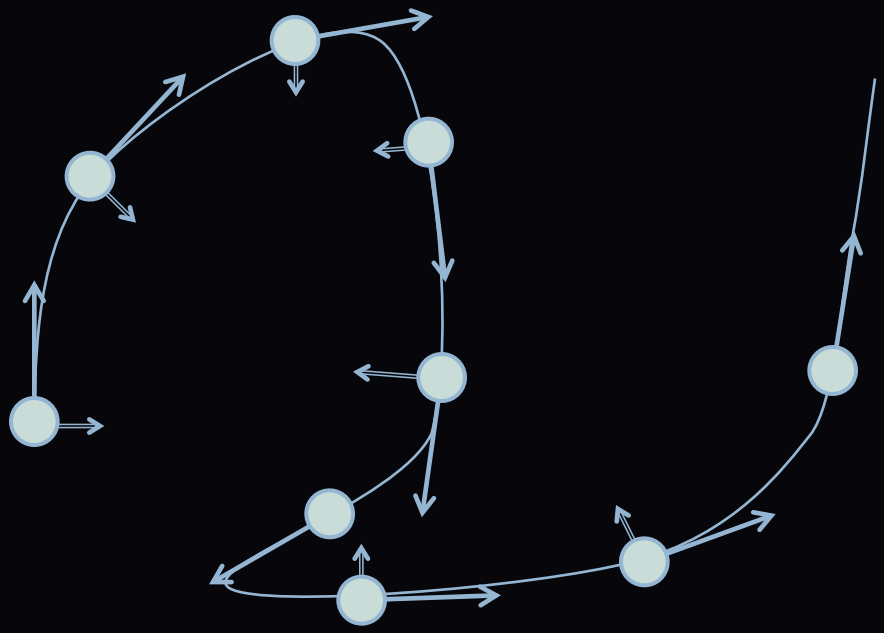
- Note that if this extrapolation is true you never need to send another event!
- It will be wrong (diverge) if acceleration changes
- BUT you can wait until it diverges a little bit before sending events
- The sender can calculate the results as if others were interpolating (a ghost), and send an update when the ghost and real position diverge

1st Order Model





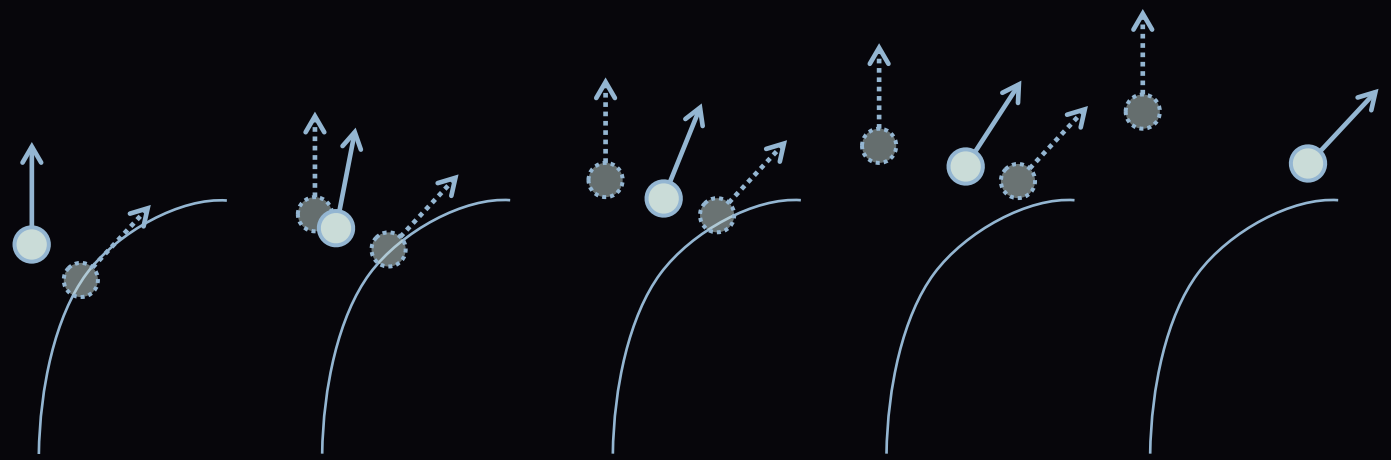
2nd Order Model



Convergence Algorithm

- When they do diverge, you don't want the receiver to just jump: smoothly interpolate back again
- This is hard:
  - Can linearly interpolate between old and new position over time, but vehicles don't linearly interpolate (e.g. could mean slipping or even going through obstacles)

Convergence Appears as Sliding Motion



Blending between the old ghost and
new ghost
over several frames

Interpolation

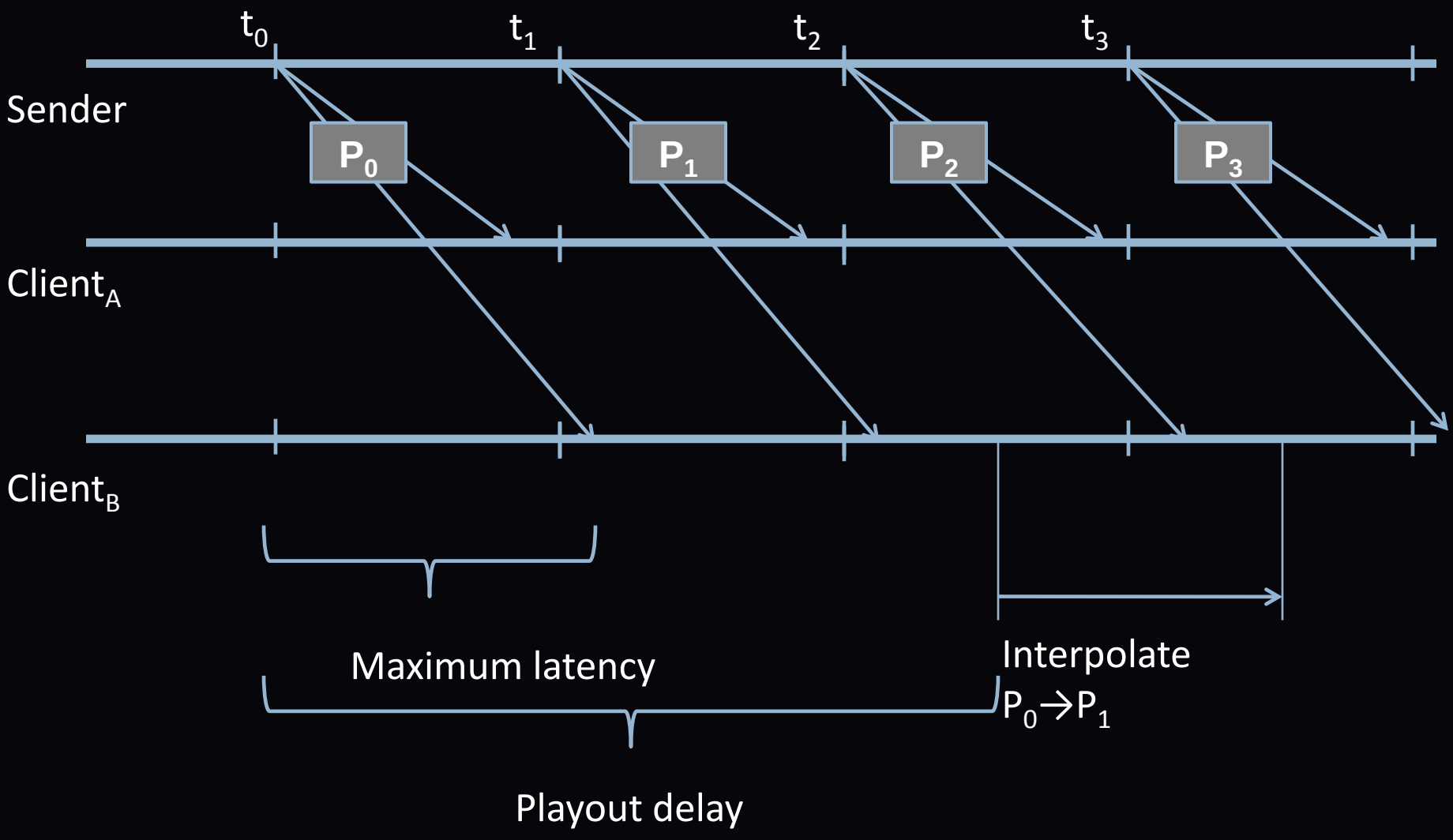
- Extrapolation is tricky, so why not just interpolate?
- Just delay all received information until there are two messages, and interpolate between them
- Only adds delay equal to the time between sending packets

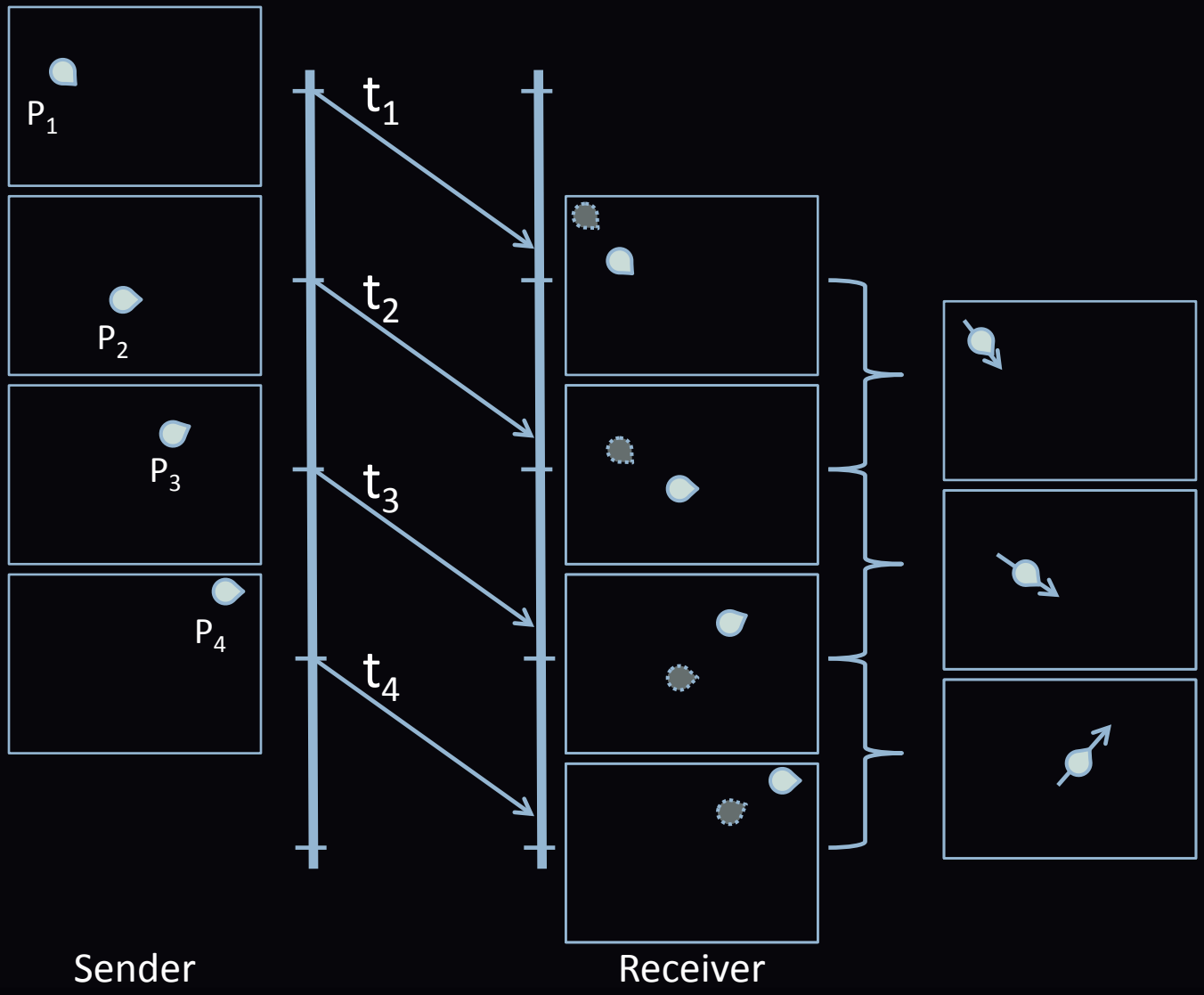
Interpolation & Playout Delays

SIGGRAPHASIA2011 HONG KONG

- Extrapolation is tricky, so why not just interpolate?
- Just delay all received information until there are two messages, and interpolate between them
- Note that jitter is not uniform, you need to be conservative about how long to wait (if a packet is late you have no more information to interpolate, so the object freezes)
- NVEs and NGs thus sometimes use a *playout delay*
- Note that if you use a playout delay on the clients own input, then all clients will see roughly the same thing at the same time!

Playout Delay

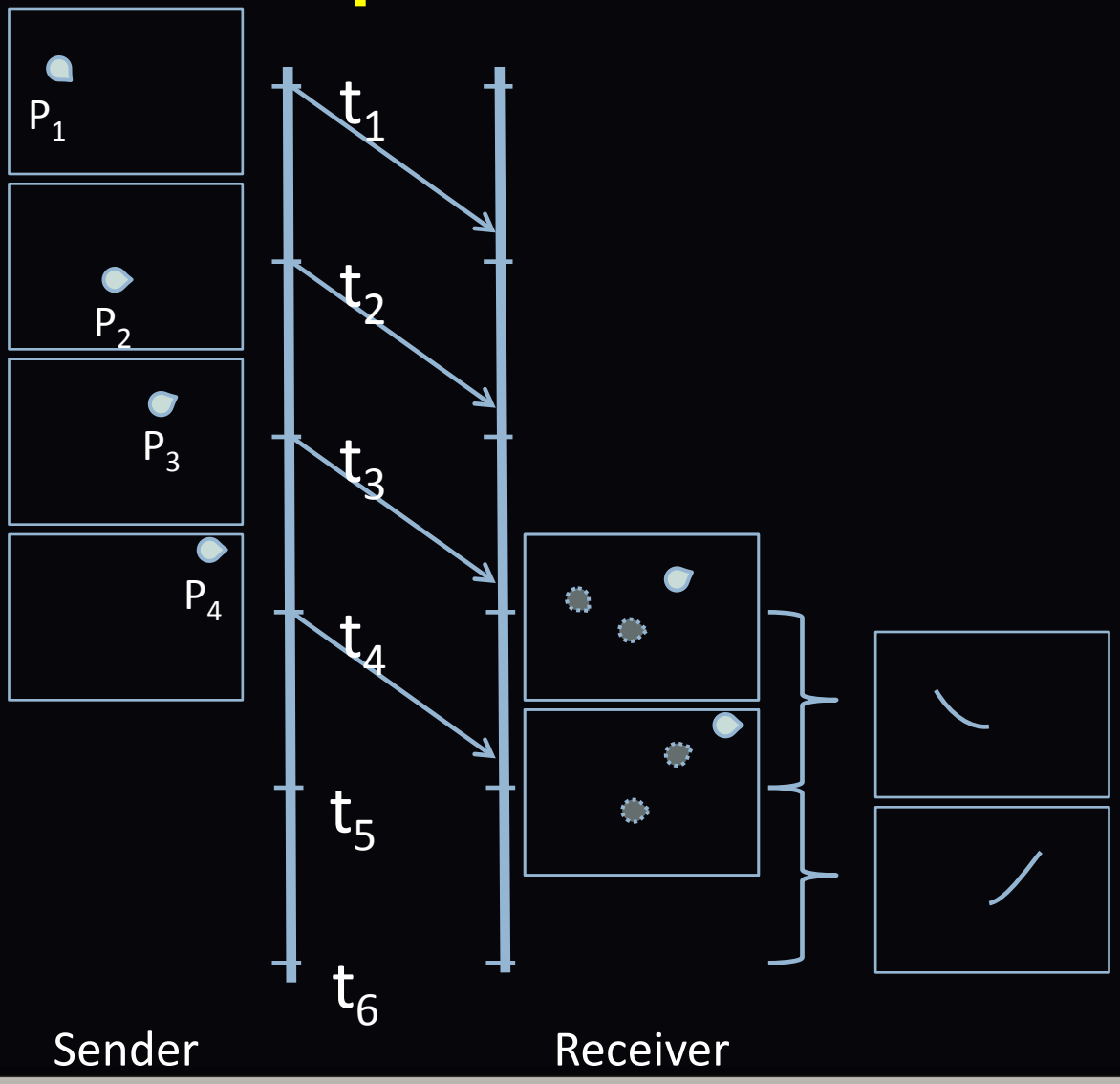



P1
P2
P3
P4
t1
t2
t3
t4
Sender
Receiver

Non-Linear Interpolation

- Need to consider several aspects
- Object movement is not linear, so could use quadric, cubic, etc. by keeping three or more updates
- Note that this causes more delay
- However, if update rate is fast, the trade off is that movement is apparently a lot smoother

Non-Linear Interpolation



Summary

- You can’t beat latency, so you need to deal with the consequences
- Over LAN you can just do a lock-step or simple synchronisation scheme
  - Server can calculate all behaviours
- Over a WAN you can’t live with the implied delays, so its comes to use optimistic schemes
- Alongside that, one might delay playouts and interpolate historic events to ensure that every site see a similar state at the same time.

Introduction to Networked Graphics

Part 4 of 5: Bandwidth Management & Scalability

Overview

- Goal:
  - To explain how bandwidth limits cause scalability problems. In non-trivial environments its simply not possible to communicate all states to all parties.

- Topics:
  - Management of awareness
  - Interest specification
  - Server partitioning

Interest Specification

- Users are not omniscient beings and thus they can't be interested in every event in a non-trivial scene
  - Plausibility needs to be maintained
- Systems thus model the user's awareness so that they can only deliver a conservative approximation to the necessary events so that the user's illusion of a shared virtual environment is maintained

Awareness Categories

- Primary awareness
  - Those users you are collaborating with
  - Typically near by, typically highest bandwidth available
- Secondary awareness
  - Those users that you might see in the distance
  - Can in principle interact with them within a few seconds by movement
- Tertiary awareness
  - All other users accessible from same system (e.g. by teleporting to them)

System Goals

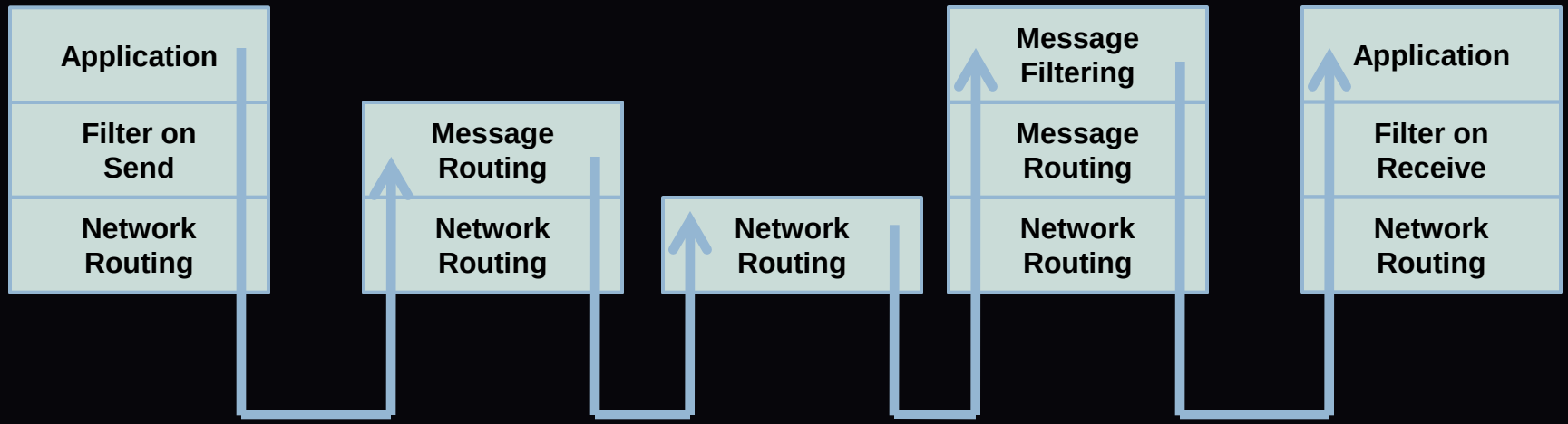
- Attempt to keep
  - overall system utilization to a manageable level
  - client inbound bandwidth at a manageable level
  - client outbound bandwidth to a manageable level

- To do this
  - Have clients discard received information
  - Have the system manage awareness
  - Have clients generate information at different levels of detail

Managing Awareness

- A complex distributed problem
- Users' expressions of interest in receiving information balanced against system's and other clients' capabilities
- Awareness scheme is partly dependent on the networking architecture, but most awareness management schemes can be applied to different architectures
- Spatial layout is the primary moderating factor on awareness

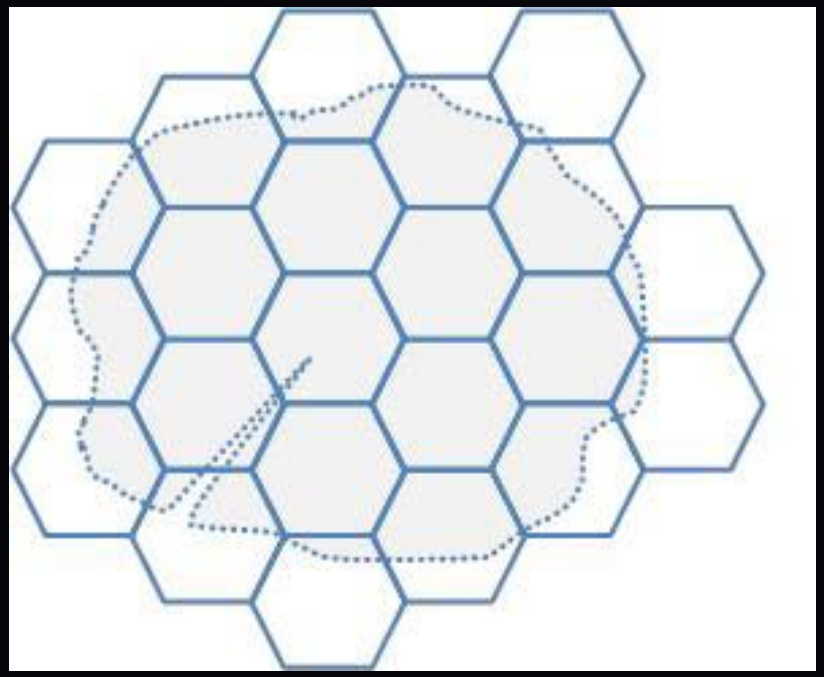
Filtering traffic



Spatial Partitions

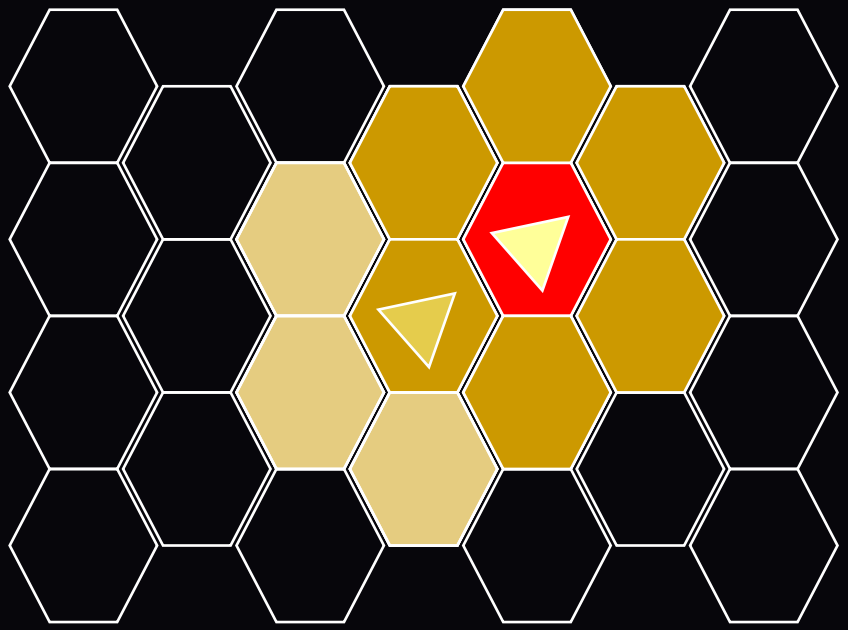
- Global Partitions
  - Static Grid
  - Hierarchical Grid
  - Locales
- Local Partitions
  - Aura / nearest neighbours
  - Visibility

Global Partitions: Static Cells



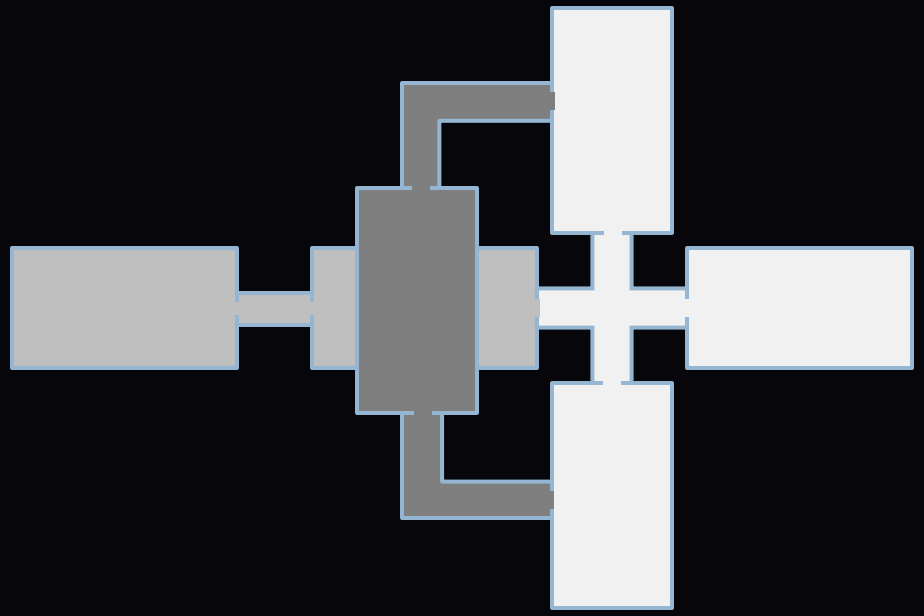
- A static partition in to regular cells
- Players only communicate with other players in the same cell

Global Partitions: Static Cells



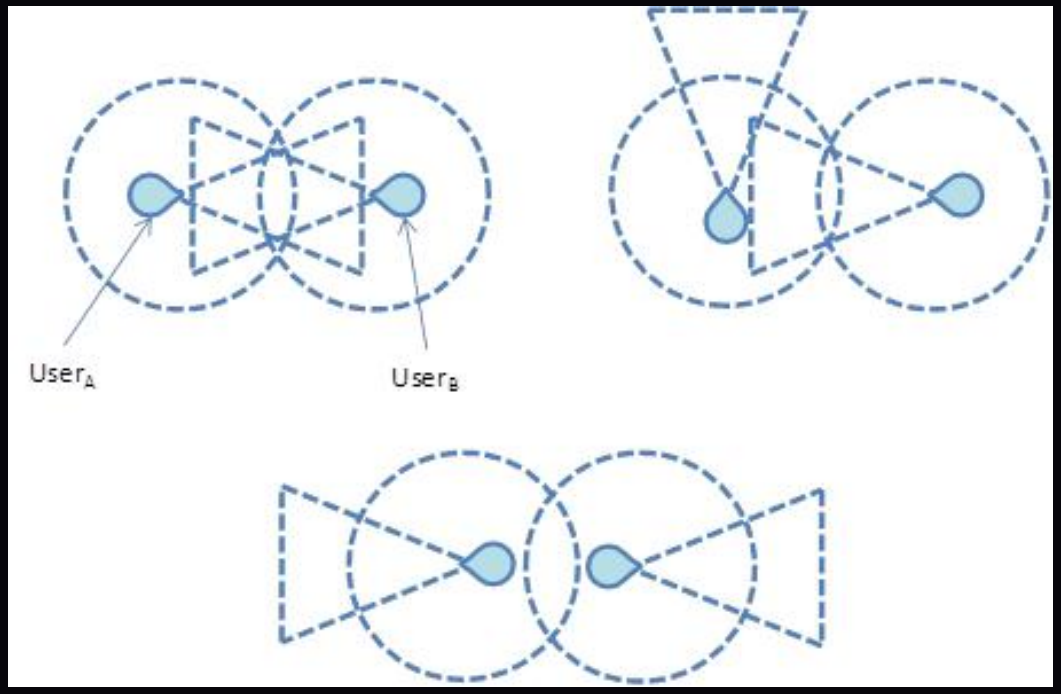
- A slightly more sophisticated partitioning
- Each player receives information from 7 nearest cells
- As they move they change the cells they receive from
- No longer abrupt changes across borders

Global Partitions: Irregular



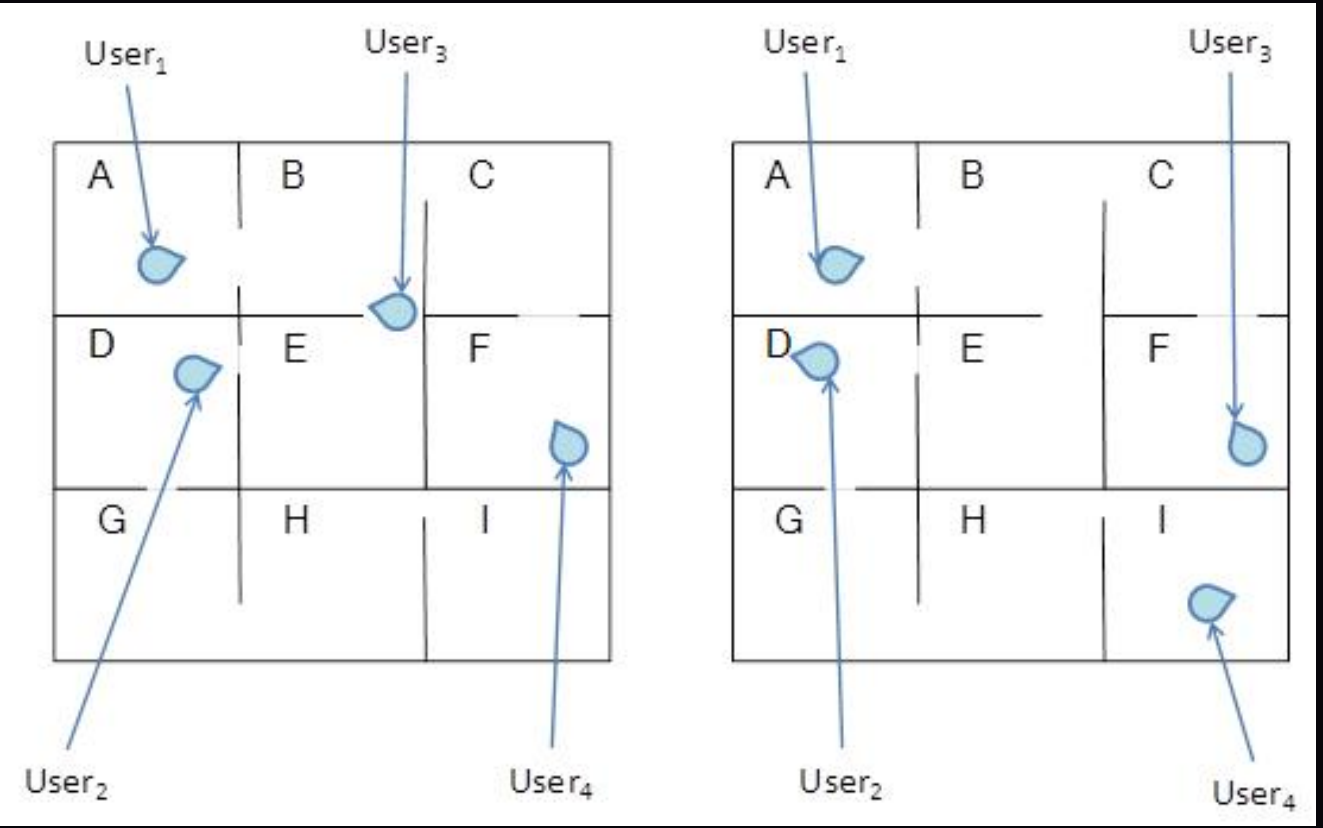
Two irregular partitionings

Spatial Partitions: Auras / Nearest Neighbours

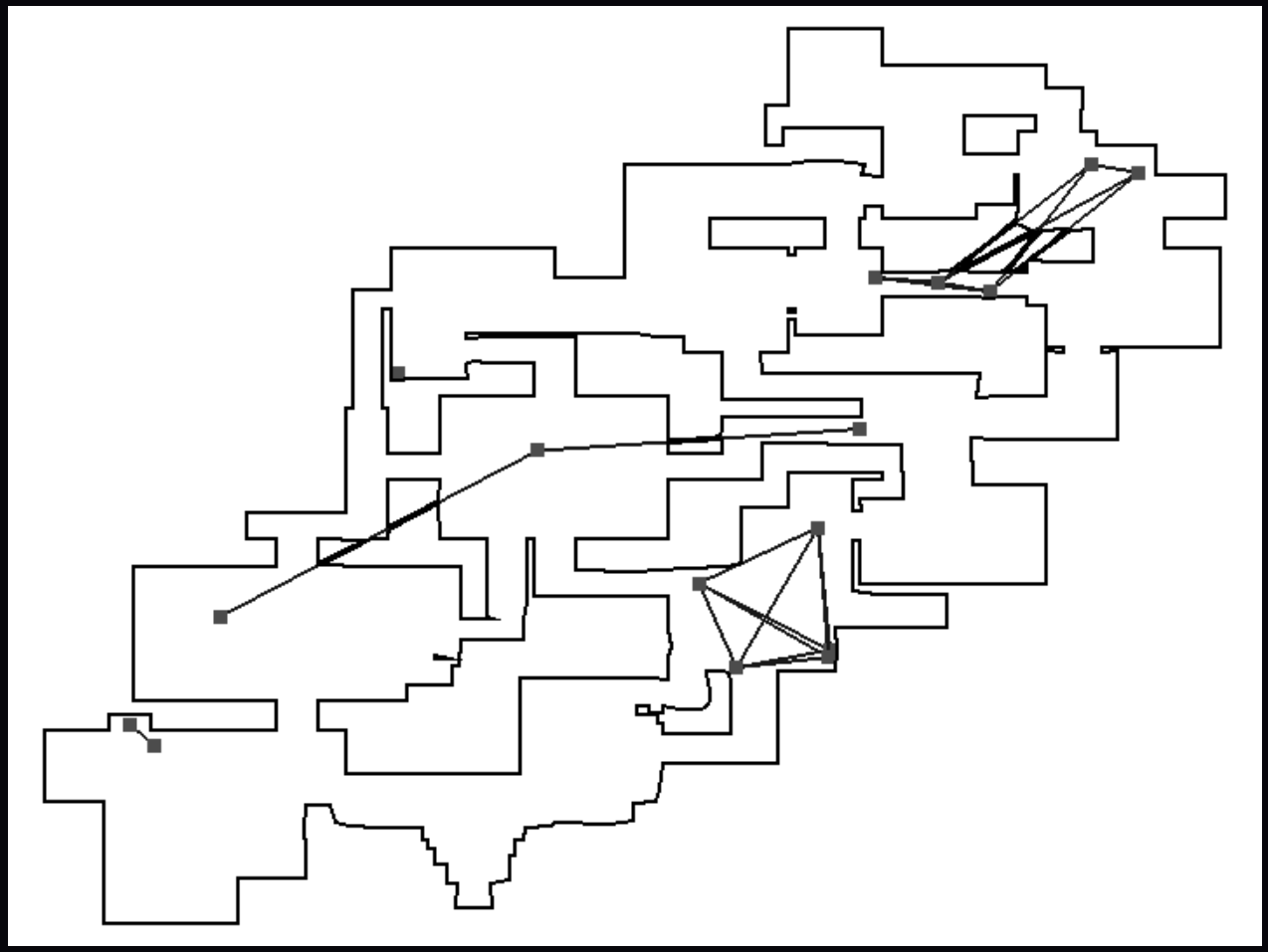


- Aura focus nimbus model from Benford, Greenhalgh, et al.
- Network connections are set up if users are close to each other and “looking” or “listening” in their direction.

Spatial Partitions: Local Visibility



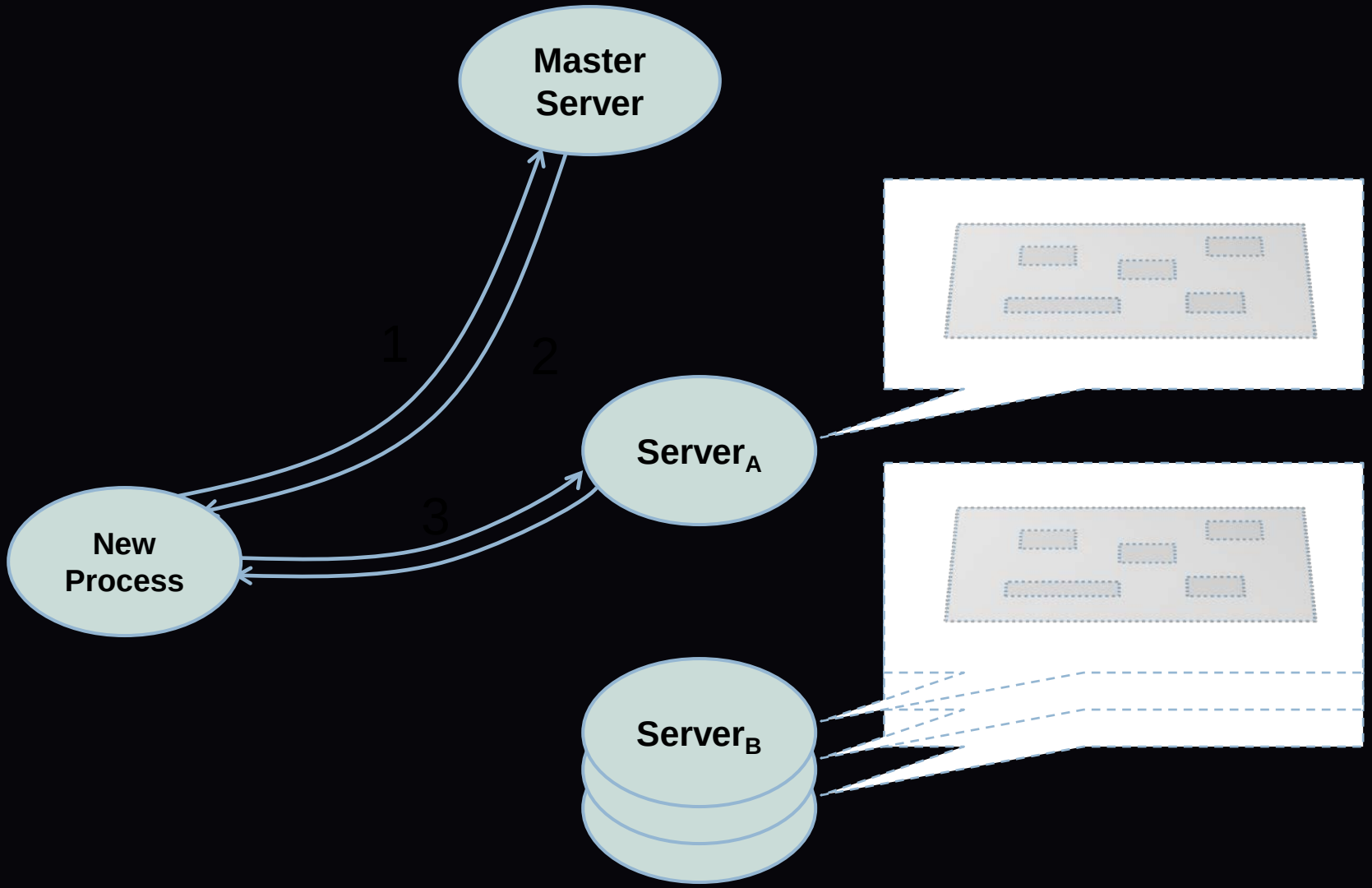
Spatial Partitions: Local Visibility



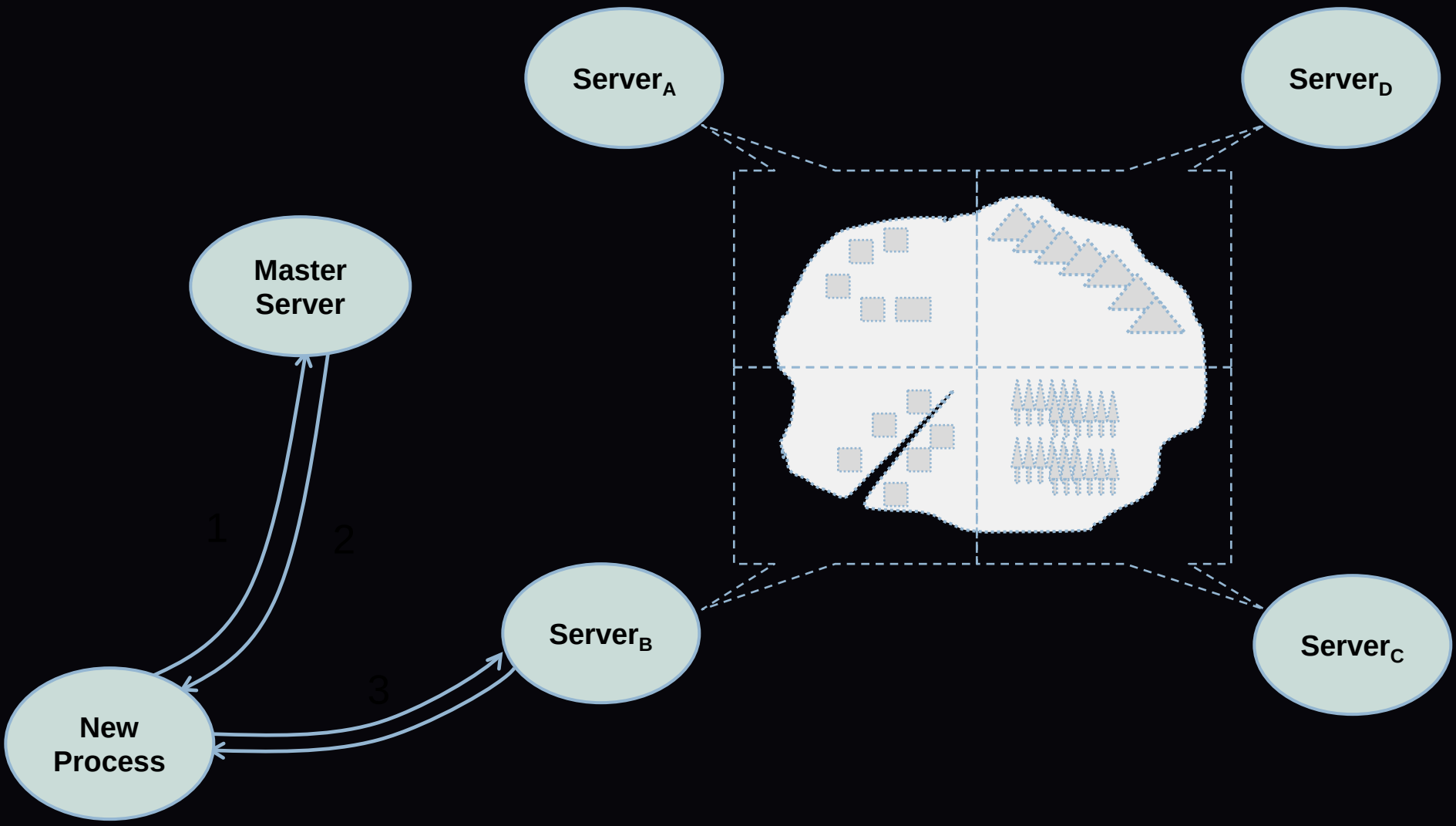
Practical Systems

- A system such as Second Life™ utilizes a regular grid layout with one server per region
  - Regions are laid out on a mostly-contiguous map
- However is a game session, far too many players want to access a specific game content
- A game *shard* is a complete copy of a system, you connect to one system and see one player cohort
- A game *instance* is similar, but is replication of a particular area (e.g. dungeon) to support one group of players within a cohort. Often created on demand.

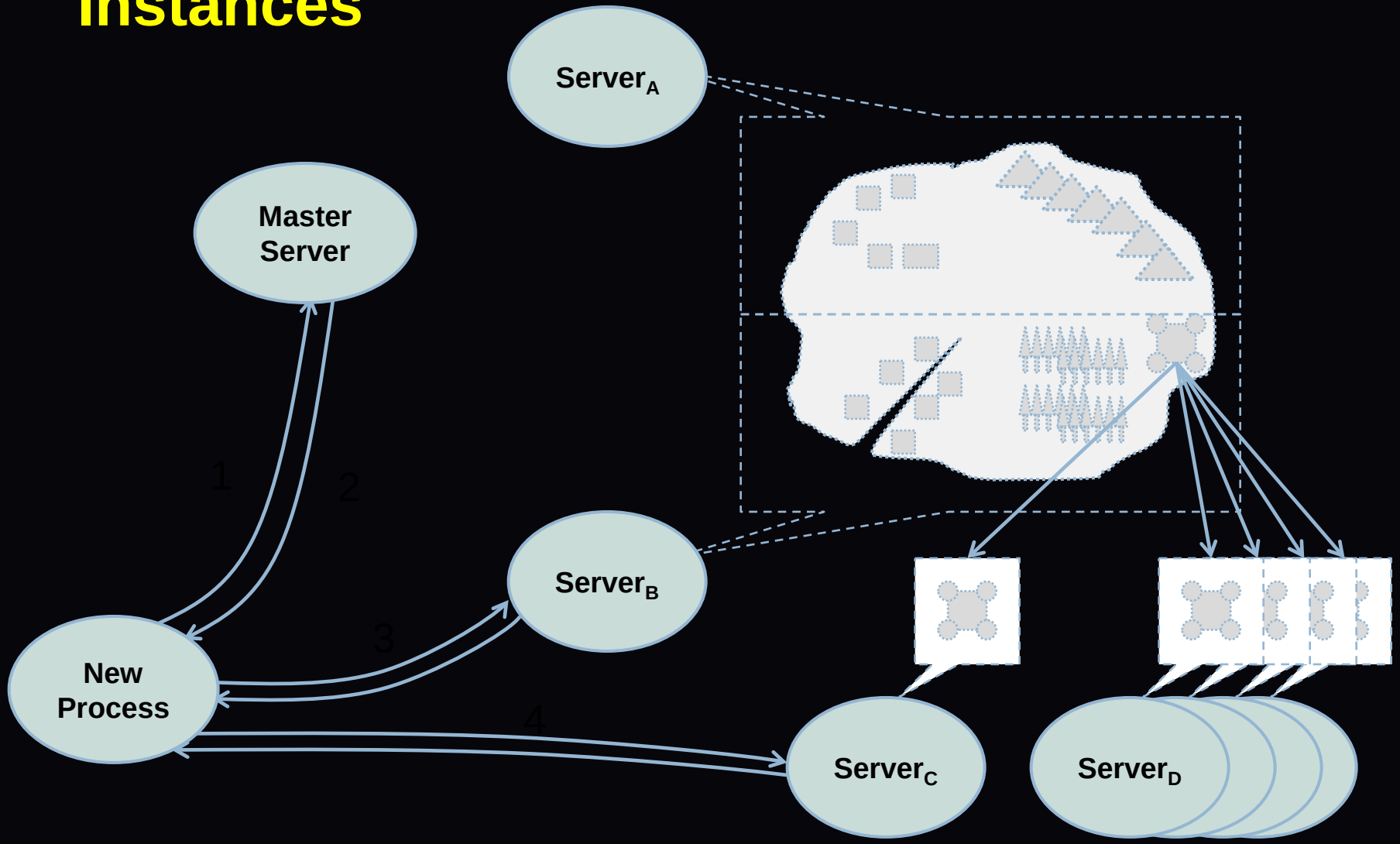
Game Shards



Game Regions



Game Regions & Instances



Summary

- Scalability depends on a choice of awareness mechanism
  - Requires a logical scalability mechanism based on what is most relevant for the users
  - Needs to consider bottlenecks at several points
  - Most common strategy is to partitioning users

Introduction to Networked Graphics

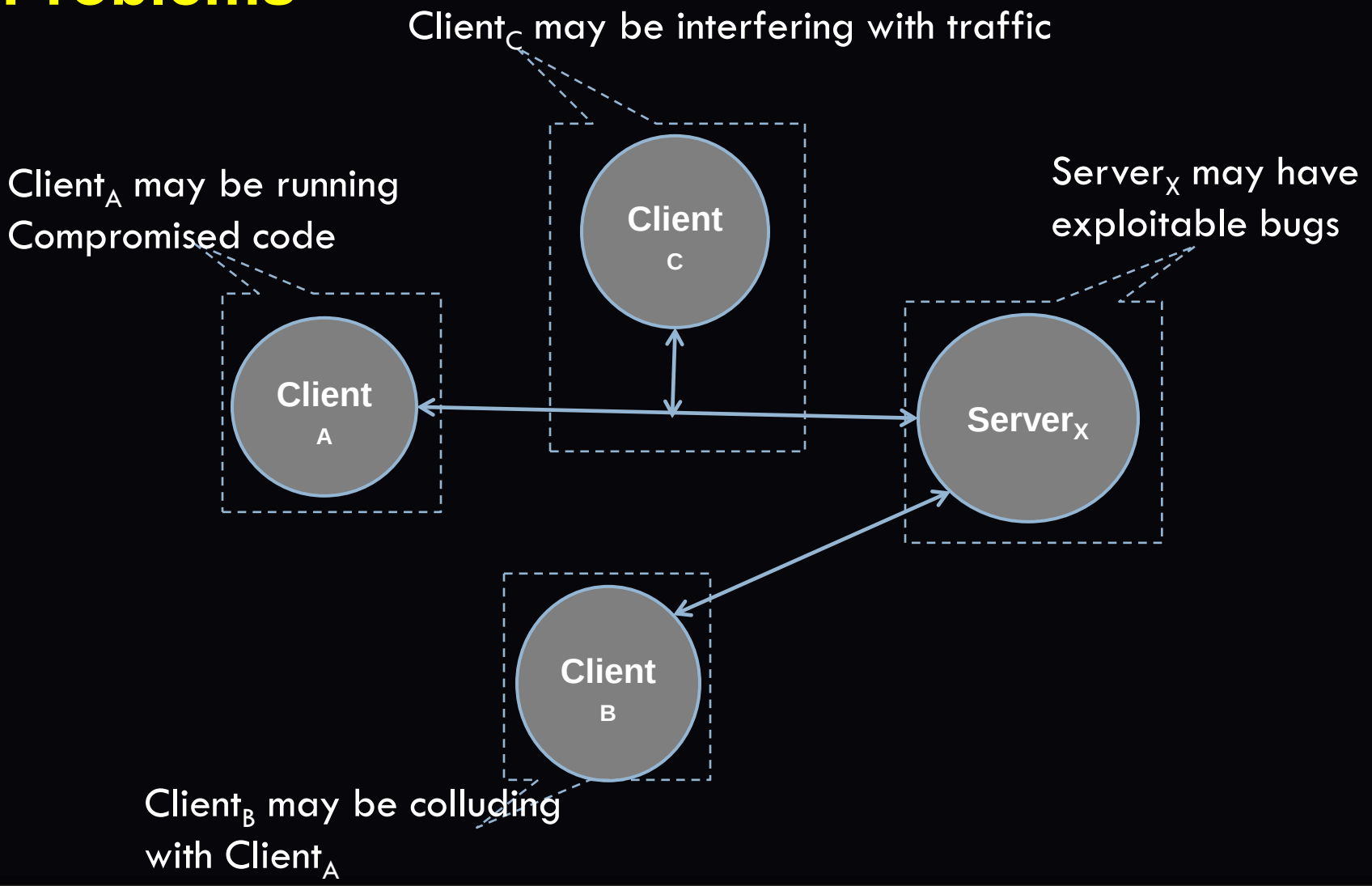
Part 5 of 5: Application Support & Recent Research

Overview

- Goal:
  - To explain some other application issues and areas of recent research.

- Topics:
  - Security and secure networks
  - Streaming
  - Cluster graphics
  - Thin clients
  - Peer to peer

Overview of Security Problems



Compromised Clients

- A pervasive problem in gaming
  - E.G. notable problems with PSNet games after the PS3 master key was found allowing modified code on the PS3
- For console gaming, hardware vendors try to lock down the hardware so only verified programs can run
- For PC gaming, various detection techniques such as PunkBuster that detect malicious software
  - Countermeasure are typically ahead of amateur cheats but not professional cheats

Traffic Interference

- Once data is on the network it is public
- Various attacks
  - Packet injection
  - Packet hiding
  - Latency asymmetry
- Some are mitigated by secure networks
  - Some servers specifically support secur

Exploitable Server

- Users need to trust server, user hosted games are not accepted for ranking tournaments or cash games
- Server might be have a loophole
  - E.G. Dupe bugs
- Denial of service attack

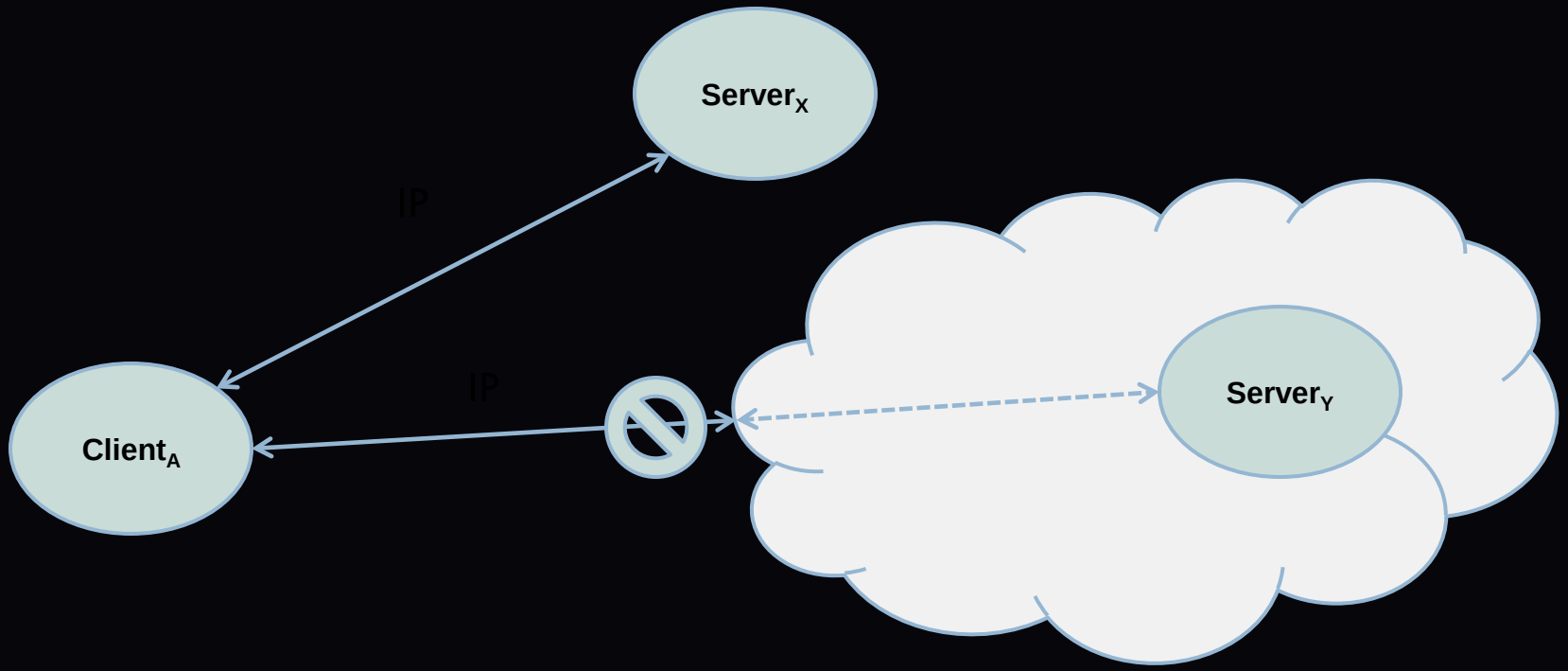
User Collusion

- A very difficult social situation to counter
  - E.G. Chip dumping
- With this and all other security problems *monitoring* of exceptions is important
  - Players being too skillful
  - Unlikely plays
  - Game inventory inflation

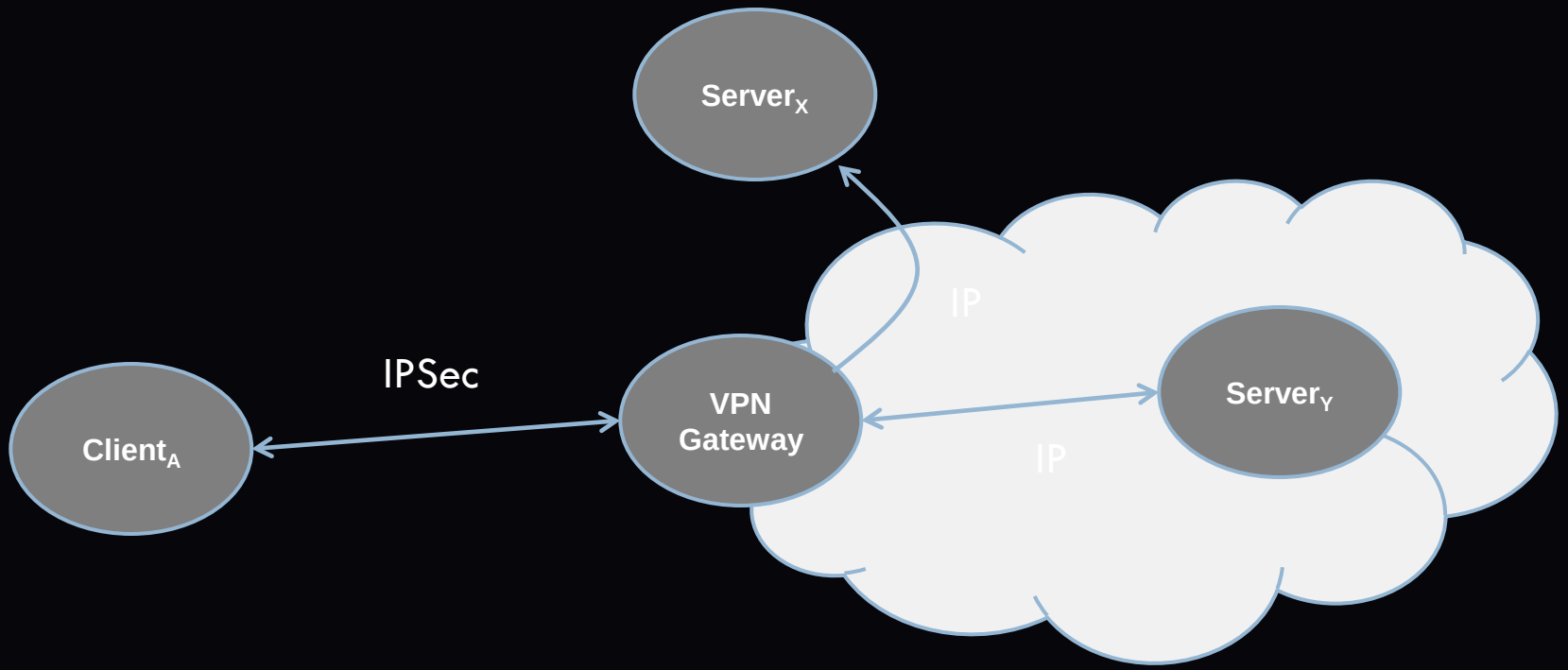
Virtual Private Networks

- Now very common for corporations and universities
- Three reasons
  - Protection of internal services
  - Giving a different “appearance” to the outside world (e.g. ACM Digital Library)
  - Security of access from anywhere (no need to trust local network)
- The very easiest way to protect a NVE or NG is to require someone go on a trusted VPN first
  - Incurs latency/bandwidth overhead of routing all information to the VPN access point first

Virtual Private Networks (VPNs)



VPNs and IPSec



Different Uses of Streaming

- Streaming Protocols
- Streaming Animations
- Streaming Geometry (i.e. incremental download)

Streaming Protocols

- Audio/video transport is well developed on the Internet
- However “well developed” means lots of competing solutions
- Several plug and play libraries
- Real-Time Protocol an extension of UDP to support streaming (though not all streaming protocols use it)
- Can get RTP compliant libraries which enables streaming and receiving
  - E.G. AccessGrid, some VoIP solutions

Real-Time Protocol

Bits	0 15		16 31
0-31	Version, config, flags	Payload Type	Sequence Number
32-63	Timestamp		
64-95	Synchronisation Source (SSRC) Identifier		
96+	Contributing Source (CSRC) Identifiers (Optional)		
96+	Header Extensions (Optional)		
96+	Payload Header		
128+	Payload Data		

RTP Payloads

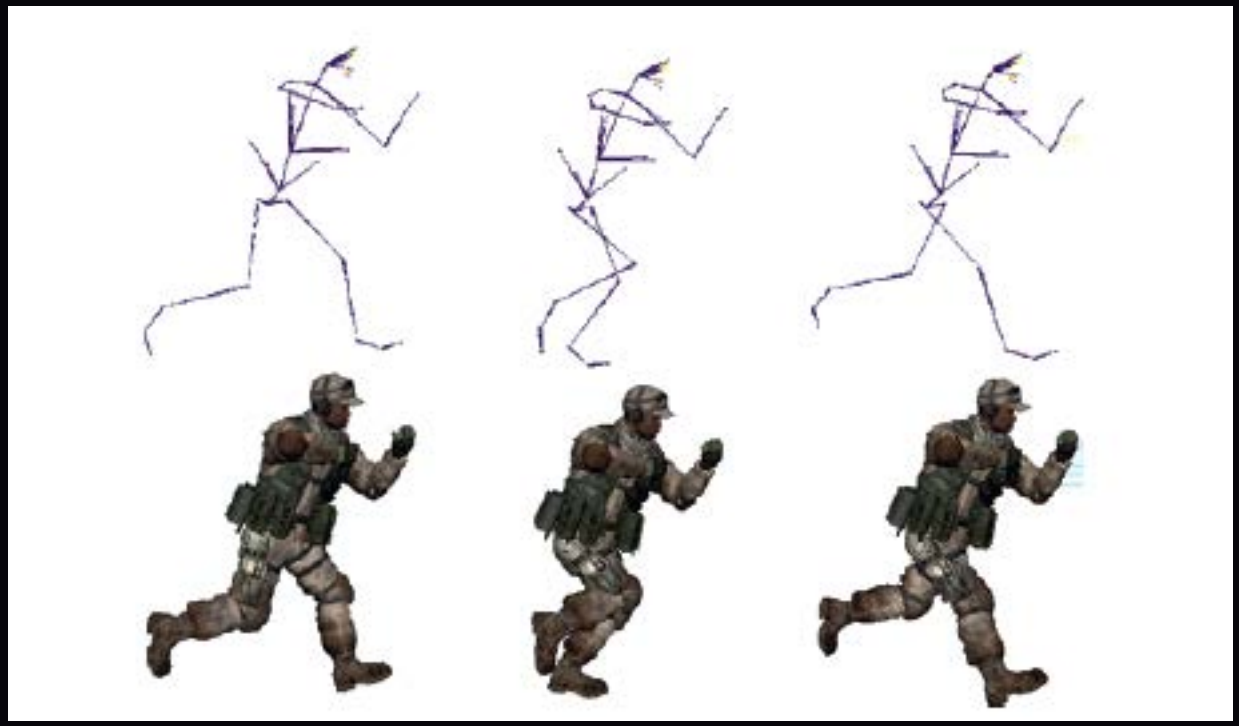
Table 13.1 Some of the Potential RTP Payloads

Description	Specification (RFC)	Type Num	Format
ITU G.711 μ-law audio	1890	0	AUDIO/PCMU
GSM full-rate audio	1890	3	AUDIO/GSM
ITU G.711 A-law audio	1890	8	AUDIO/PCMA
PureVoice QCELP audio	2658	12	AUDIO/QCELP
MPEG Audio (e.g. MP3)	2250	14	AUDIO/MPA
Motion JPEG video	2435	26	VIDEO/JPEG
ITU H.261 video	2032	31	VIDEO/H261
MPEG I/II video	2250	32	VIDEO/MPV

Streaming Animations

- We have already looked at streaming positions and orientations of objects
- However, a large class of objects are humans or animals (or aliens) which deform
- Typically modeled from the graphics side as a skeleton
- Animation is controlled by indicating which *motion* the character is in and the *keyframe* in that motion
- Because motion is continuous (e.g. motion capture) information might only need to be sent > 1s

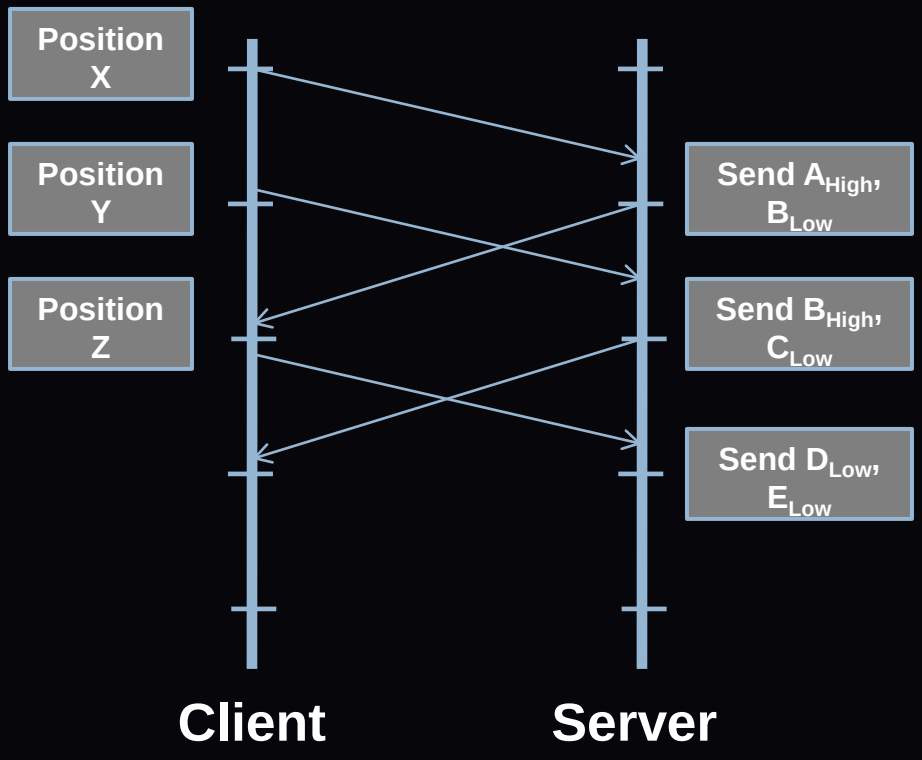
Examples of Keyframe Animation



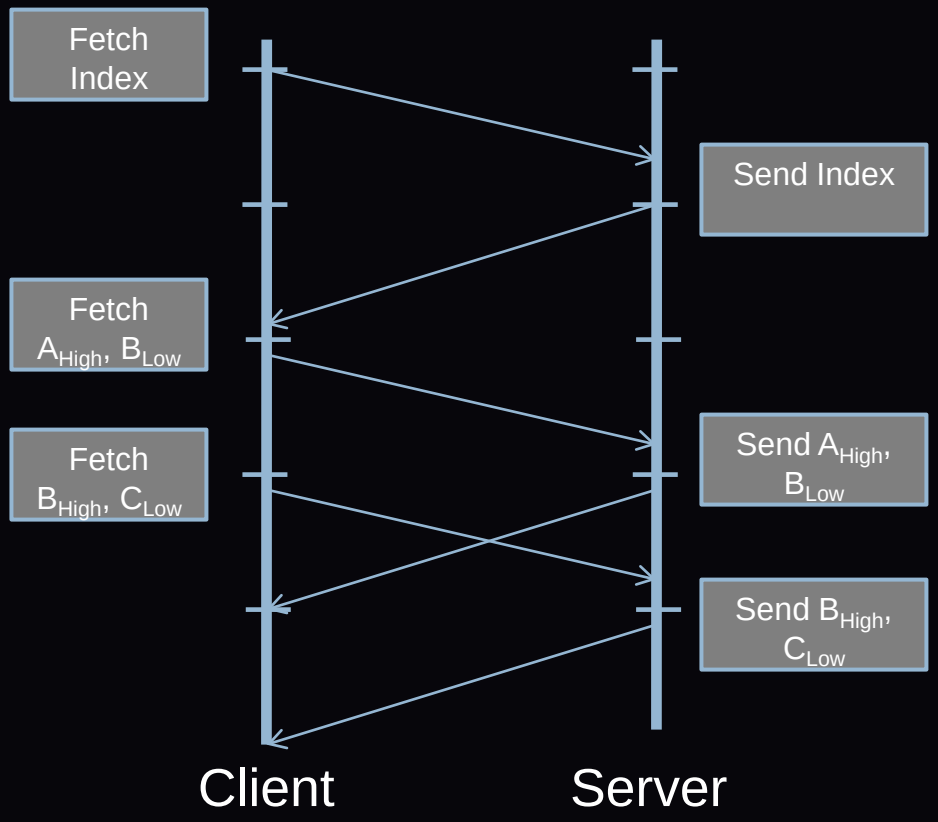
Streaming Geometry

- Many NVEs use very large worlds which need to be downloaded because user modifiable or just vast
- System needs to determine which parts of the models should be transferred
- Typically done in a *priority order* from the viewpoint of the client, e.g. in increasing distance order
- Two ways of doing this
  - Client-pull
  - Server-push

Server Push


Position X
Position Y
Position Z
Send A_{High}, B_{Low}
Send B_{High}, C_{Low}
Send D_{Low}, E_{Low}
Client
Server

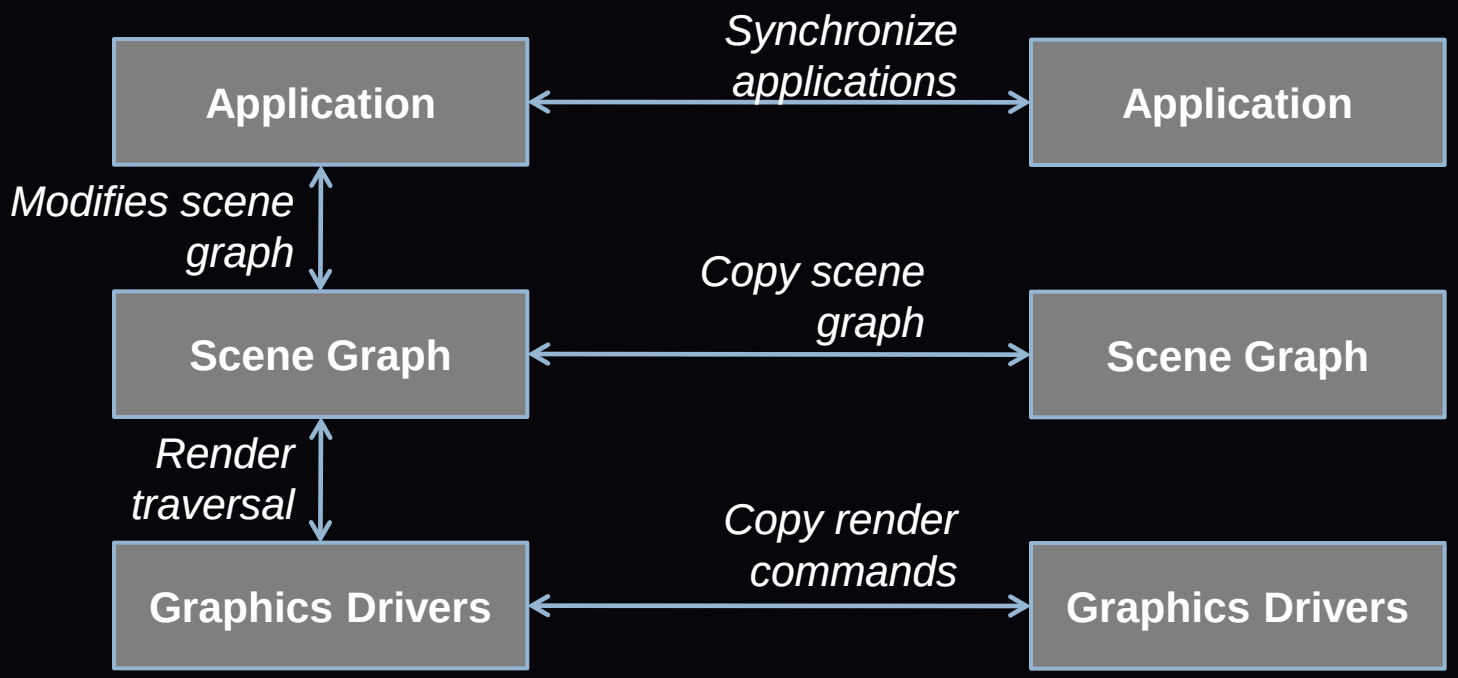
Client Pull


Fetch Index
Send Index
Fetch A_{High}, B_{Low}
Send A_{High}, B_{Low}
Fetch B_{High}, C_{Low}
Send B_{High}, C_{Low}
Client
Server

Clusters

- Cluster graphics is a particular concern of Virtual Reality system designers
- One GPU card generates one or two video to get maximum throughput, but we might need 4+ displays
- Need to synchronize graphics at two levels
  - Synchronize graphics state on input to rendering
  - Need to synchronize video output

Layers of Sharing Graphics



Tools

- Copy render commands
  - E.G. Chromium – stream OpenGL commands over TCP/Ethernet, or other non-IP-based interconnects
- Copy scene graph
  - E.G. OpenSG – stream an edit change list for a scene-graph
- Synchronize applications
  - E.G. VRJuggler – isolate all input in to one (or more) C++ classes that can serialize themselves to the network, stream the resulting serializations.

Thin Clients

- Might be considered “backwards” but graphics architectures go in circles, so why not networked graphics architectures
- Render the graphics on a server, stream the results as video
- Recent consumer examples: OnLive, OToy, GaiKai
- However many OS vendors have such a functionality for supporting thin clients over LANs

Thin Clients

- Very small installable on client, client doesn't need to be high-powered (hence thin client)
- Stream to server your controller input
- Stream back video (e.g. 720p from OnLive)
- Server runs both game client and game server (actual architectures not revealed)

Thin Client Pros and Cons

- Pros
  - Very small installable (e.g. only Flash for GaiKai)
  - Thin client can be low power (e.g. Netbook)
  - No need to download/install very large game assets
- Cons
  - Latency
  - Constant high bandwidth use compared to normal game network traffic

Peer to Peer

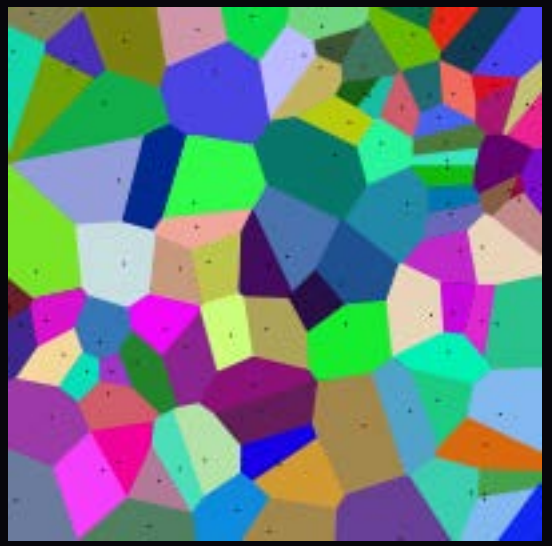
- A live challenge: how can peer to peer networks scale up to very large numbers
- Key to this is how to distribute awareness management
- A secondary issue is how to “bootstrap”: how does a user find their local users?

Larger Peer to Peer Context

- Enormous work in networking community on generic large scale peer to peer databases
- Key technologies
  - *Distributed hash tables*: a way of storing data sets across multiple hosts but ensuring fast (O(logN)) access to any data item
  - *Application-level routing:* a mechanism for supporting group peer to peer communication without any underlying network support

Within a NVE Context

- Very active line of research
- For example, can one maintain a set of closest peers with something similar to a Voronoi Tessellation?
- If peers can identify their Voronoi Cell, they can identify their neighbours.
- New clients can walk the cells to get to find their true neighbours



Summary

- Plenty of tools and options to support your NG or NVE project
- Security is a big challenge if you can’t get your users on to a VPN
- Other facilities require more infrastructure and are very domain specific
- Plenty of research issues: thin clients being a wild card at the moment

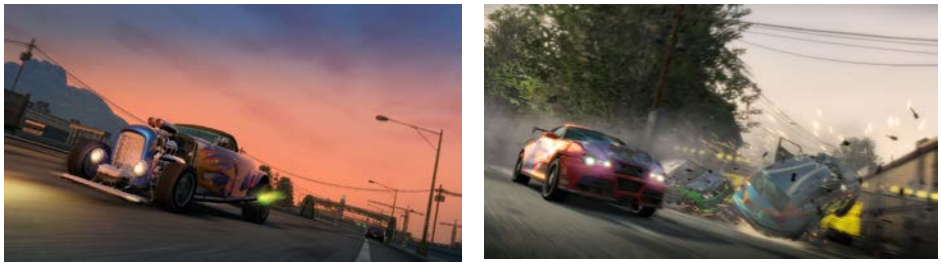
Case Study: Burnout™ Paradise

The Burnout™ series of racing games is developed by Criterion Games, who were acquired by Electronic Arts in 2004. The Burnout™ games are renowned for their focus on high-risk and high-speed driving and spectacular crashes. The games have garnered very positive critical acclaim and a number of awards. Burnout™ Paradise is the fifth game in the series. It was originally released for Xbox 360 and PlayStation 3 in January 2008. Since then several online updates have been made available, and a PC version has been released.

Whilst the first four games in the series had a traditional format based on races or other events that could be selected from a front-end menu, the Burnout™ Paradise game format is very novel. Players drive around a large open world (Paradise City) and start races and events by stopping at traffic lights and revving their engines. The race then starts from that location. This has two quite novel implications: players can access most events right from their entry to the game and the player can turn up in any car that they have access to, and finally, and perhaps most controversially.

Much of the gameplay focuses on the causing of crashes. Part of the fun and reward is "taking down" other players' vehicles or non-player vehicles by causing them to crash. There are several types of event, including straight-forward races, road rage where the player must takedown a target number of vehicle, marked man where the player must avoid being taken down, stunt runs which require the player to sequence together jumps and stunts without crashing, and burning routes which are timed challenges using specific vehicles. All of these event types, with the exception of the burning routes are available in multiplayer mode. In addition there is a sequence of challenges that require cooperation or competition amongst the players.

Multiplayer mode, called Online Freeburn in the game, supports eight players at once. The player can drop in and out of multiplayer mode whilst driving around Paradise City. They enter the multiplayer mode in the same location, and may be in the thick of the action, or on the other side of the city. Also, and importantly, there may or may not be an event or challenge already in progress. If there is, then the new player doesn't join the event, but can drive around and observe until the next event starts.



SCREENSHOTS FROM THE PC VERSION OF BURNOUT™ PARADISE

Car Mechanics

The basics of the multiplayer mode are the distribution of the players' cars. Burnout™ Paradise uses a peer-to-peer model, where each player sends updates to each other player. The game runs at a fixed rate (50 or 60Hz), and the player sends out updates in a round-robin manner, in that each frame they send an update to at most one of the other players. They actually send approximately 6 packets to each other player each second. This means that each client must animate the other players' cars based on updates 160ms apart. Latency between clients becomes problematic when it gets large, for example at over 500ms, but this is very rarely seen.

Burnout™ Paradise involves racing at speeds of up to 200mph and it is important to preserve the impression of racing neck and neck through complex landscapes. However the update rate poses a challenge: if packets arrive every 160ms: the vehicles may have travelled 15 meters between updates.

Furthermore the latency of the link means that for every ms latency, the vehicle could have travelled almost 0.1m in the game. Early prototypes of the game solved this problem by playing out game state at a fixed play out delay, meaning that even in the presence of dropped updates, the simulation was always consistent. However, for Burnout™ Paradise, with its emphasis on close racing and trying to push rivals in to the scenery, this wasn't acceptable. Thus each client extrapolates the last known state of the other cars. Each player update message contains the car position and velocity, but also the current control input. (acceleration, braking, drifting, etc.). If the control input doesn't change, then the extrapolation will be exact, however control input will frequently change during these periods. Thus whilst extrapolating the state of the other cars, the client enforces some local visual and physics consistency: player cars are moved around static scenery and traffic. Each client is completely authoritative about its own car, and thus other clients can't force it to crash in to their own car or other scenery. Each client takes the forces generated by the other cars and integrates them. This means that if you nudge another vehicle you may notice their response is slightly delayed, but your own car's response is actually under a physics simulation with car roll and momentum, so your own car's response is instantaneous.

Extrapolation of last known state may lead to consistency repairs later: for example the local extrapolation may give the impression that the other player missed an obstacle, but actually they hit it. Thus the collision would have happened in the past at the local side. The local client, given the time in the past when the collision did happen, can accurately calculate where the other player car now is. The system doesn't try to converge the extrapolated non-collision state and the new state with collision, it just switches to the latter. In practice, cars carry a lot of forward momentum through crashes, crashes are so complex and the game moves so quickly, that this isn't noticed too frequently.

Paradise City is full of traffic, with thousands of vehicles touring the city or parked on the road-side. Any traffic is potentially collideable. Furthermore, collisions and crashes are under the control of a physics simulation. At first glance this means that enormous amounts of state must be shared between the players. However, two implementation strategies avoid the need for large state synchronization: the traffic is actually deterministic depending on the game time, and the physics engine is completely deterministic. Thus as each player drives around the city, the non-player traffic is automatically consistent, as long as the players' game time clocks are synchronized. To synchronize any non-player vehicles that have been hit, the clients only need to send the initial collision itself, that is the velocity and direction of impact and status of the cars, and then each client can evolve the state, including further collisions. Each player is responsible for the collisions that they themselves cause with non-player traffic.

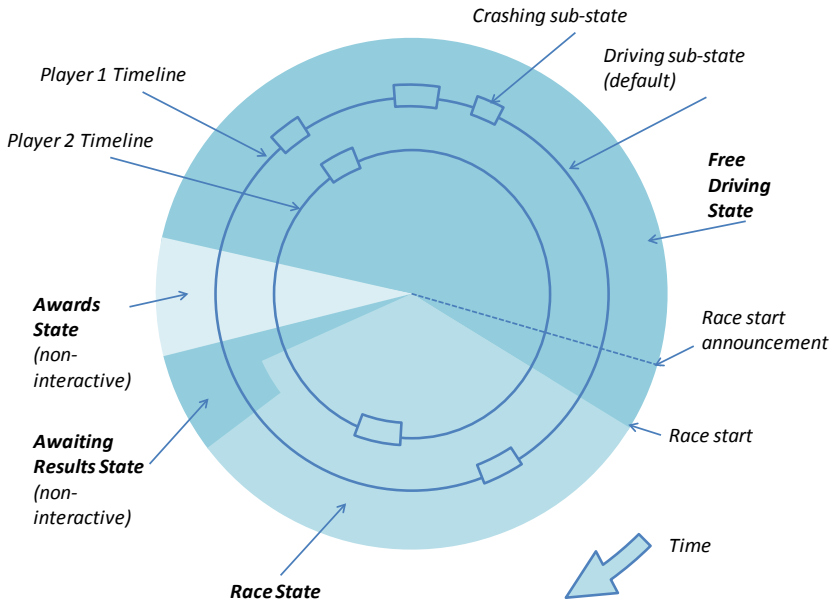
Thus in order to synchronize the main state of the game, in addition to their own car's status and control input, each frame they each player needs to send the collision descriptions (i.e. collision time and contact specification) for any collisions that they are responsible for. The player can easily be responsible for 10 collisions and thus this can comprise a large part of the data. However, it only needs to be sent once to each peer. The data sending rate of a client is thus kept to approximately 8-9kbps. The networking uses UDP because there is little point resending lost packets containing vehicle dynamics which will have changed before the transport layer triggers the resend..

Aside from the control and collisions, other items that must be sent peer to peer are car status (damage, boost, etc.), camera shots (when you takedown a rival you can send a camera shot to brag), any score multipliers and achievements, and any takedown notifications. In total there are over 50 pieces of game state that might be need synchronizing. Some things are completely unsynchronized, including the state of any collideable street furniture.

Game Phases and Time Synchronization

The majority of the network packets are concerned with creating a consistent city full of traffic. Online Freeburn provides events and challenges for players to undertake. The game thus goes through a circle of states. In building the networking reliably synchronizing the progression of these phases, and dealing with all conditions of players not getting certain information, or disappearing from the network was a big challenge. The main states are illustrated in the figure below which shows circular timelines for two players in a race situation. Both players start in the "free driving" state. The host is one nominated player who is allowed to start challenges and events. They announce the start time for the race. Once the start time is reached, each player enters the race mode by going in to an running event state This running event state, actually consists of two sub-states, "driving" and "crashing", where during driving the player has control of the car, and during crashing control is taken away. Crashing is an implicit penalty in any race as it takes a few seconds to occur before the car is put back on the road and the driving state is re-entered. In a race, once each player crosses the finish line, they enter a

"waiting results" state. Once each player crosses the finish line, or a timeout occurs, the state moves on to "awards" state where a ranking is made and points awarded. Then the players go back to the "free driving" state. In some events or challenges, there is no waiting results state, as soon as some event, or combination of events happen, the players can switch to the awards state. Moving these global states requires a reliable application-level protocol on top of UDP.



MAJOR AND MINOR GAME STATES IN ONLINE FREEBURN IN BURNOUT™ PARADISE. IN FREE DRIVING AND RACE STATE, THE PLAYER MAY CRASH. THE RACE STARTS A SHORT PERIOD AFTER AN ANNOUNCEMENT. THE AWAITING RESULTS STATE IS ENTERED AS THE PLAYER FINISHES. THIS STATE AND THE AWARDS STATE, ARE NON-INTERACTIVE.

It is critical to the fairness of the race is that everyone start racing at the same time and that they see each other in the correct order on the road. However it is not straightforward to ensure this. In practice the winner of a race is whoever takes the shortest time to cross the finish line, based on their local clock. It is not specifically required that everyone start at the same time, though this is highly desirable. The fact that you see the correct number of players in front of, or behind you, is actually a secondary concern in ensuring that the overall winner of the race is appropriate. This does lead to the situation where the first player to go in to the "waiting results" state may not be the winner: someone who actually started later, but who took a shorter time might bump them from the top. Obviously this violates the expectations of the users, so it is to be avoided.

Two techniques are used to ensure as tight as possible synchronization between the views. First, whilst in free driving state, all the players are synchronizing game clocks with a protocol similar to NTP (see Section 10.3.3). This means that, as discussed, the clocks of the machines can be synchronized to approaching 10ms accuracy. Despite this, clocks may get out of synchronization, and NTP-like protocols do not work over links with asymmetric timing. However, problems are reported to be infrequent. Note that NTP is not run during a race because of the concern above that the elapsed local time is the actual measure, and thus the local clock should not be altered during a race. The second technique is to make very sure that every client knows when the start time of an event is. The start time is broadcast from the game host five times, and in case there are any connectivity problems between two hosts, each peer also relays the start time to all peers. This might seem over-kill, but if a player's client fails to get this event then they must wait until the next game phase to join in, which could be in over 5 minutes time. In the case when a peer gets a start time for an event in the past, then they start immediately.

Game Hosting and Peer to Peer Networking

Multi-player games are set up via a master host list, hosted at an EA server facility in Virginia. Anyone can host a game, and the master host list is downloaded by each client who is online before they enter Online Freeburn. The host list contains some details of the game modes that the host is running at the time and the number of players currently connected to that host's game. When the player enters Online Freeburn, they then download the client lists for each hosted game. The host is only responsible for starting the events and challenges; they perform no other special networking function. If the host player becomes unavailable, then there is a host migration protocol to select another host from the other players.

Console players log on using their Xbox LIVE gamertag or PSN ID; this is provided by the console itself. PC players must use an account created on the EA server facility. This server facility is based on LAMP technology (an acronym referring to a suite of Open Source software, Linux, Apache, MySql, PHP/Perl/Python). Aside from logins and match-making, it is responsible for keeping track of event results, and also player-to-player states; the game makes a significant gameplay feature of rivalries and retribution on the road, not just within any one race, but over separate sessions. Match-making and back-end server provision was considered to be one of the biggest challenges, because, as has been found by many MMOG writer, it is very difficult to simulate all the players turning up simultaneously. The provision of an early demo helped Criterion Games iron out any problems.

Because the game is peer-to-peer, each client must retrieve an IP address for the others peers, but it can't be known if these are behind NATs or not. The system tries a version of UDP hole punching (see Section 10.6.1), but if this does not work, it reverts to using packet relaying. To support relaying, EA provides a number of relay servers on each continent, so that relaying does not add significant latency overhead. Thus a peer-to-peer connection can always be made between two hosts.