

Part 3:  
Latency  
Scalability

# Introduction to Networked Graphics

IEEE Virtual Reality 2011

Anthony Steed



- *Latency*

- - Mitigation strategies

- - Playout delays, local lag and dead reckoning

- *Scalability*

- - Management of awareness

- - Interest specification

- - Server partitioning

Latency

# Introduction to Networked Graphics

IEEE Virtual Reality 2011





- *Latency*

- - Mitigation strategies

- - Playout delays, local lag and dead reckoning



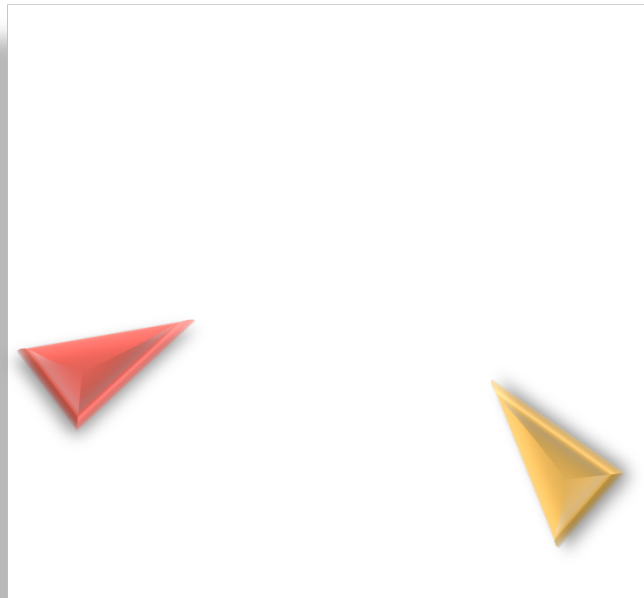
# DUMB CLIENT AND LOCKSTEP SYNCHRONISATION

# Naïve (But Usable) Algorithms

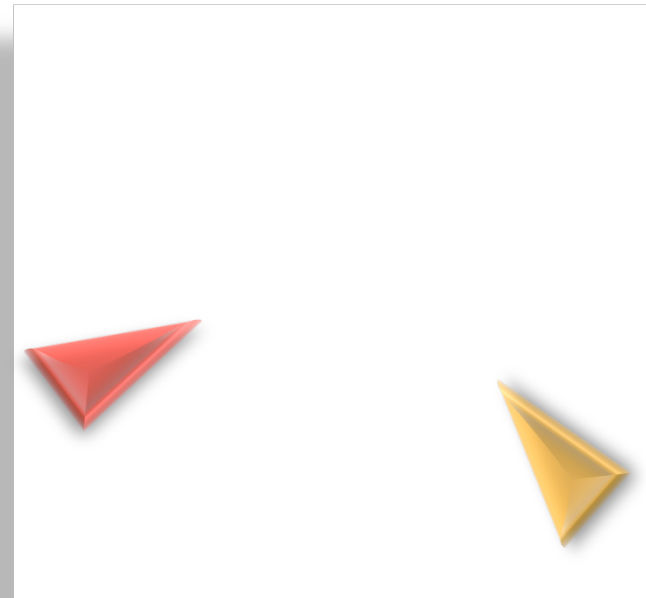


- Most naïve way to ensure consistency is to allow only one application to evolve state at once
- One application sends its state, the others wait to receive, then one proceeds
- Is a usable protocol for slow simulations, e.g. games
  - ▣ Not that slow – moves progress at the inter-client latency
- Potentially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

# Total Consistency (Alternating Execute)



Client A

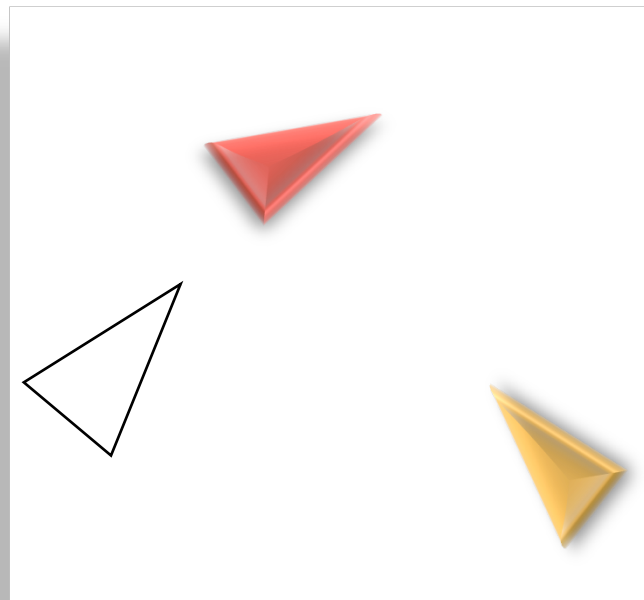


Client B

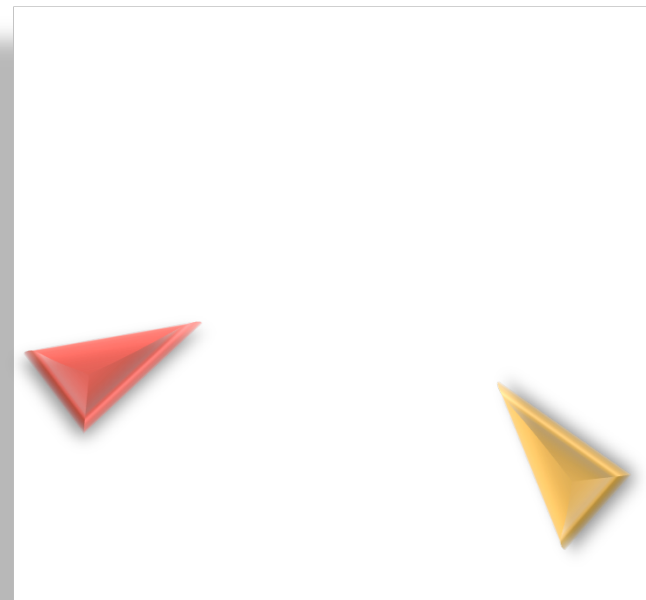
$T = t$

Acknowledge every update  
Propagation delay is 100ms

# Total Consistency (Alternating Execute)



Client A

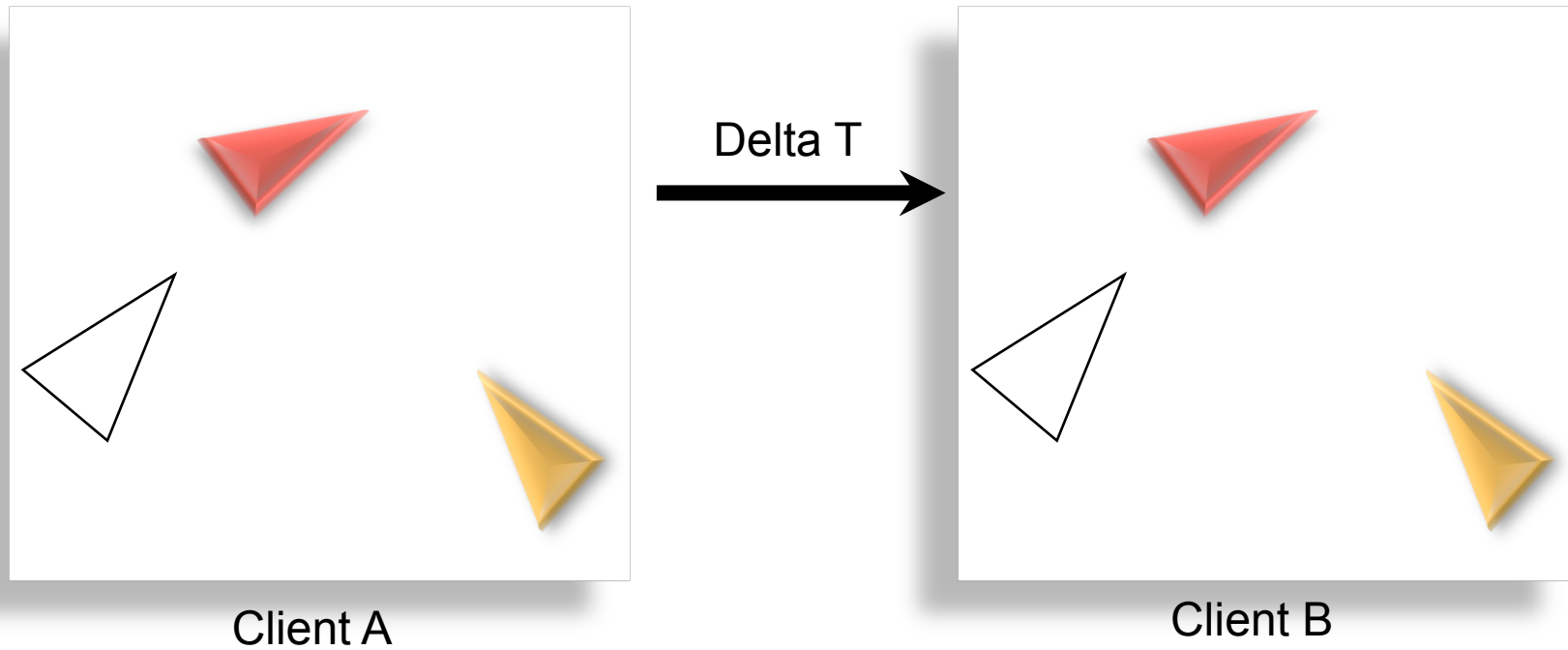


Client B

$T = t + 50\text{ms}$



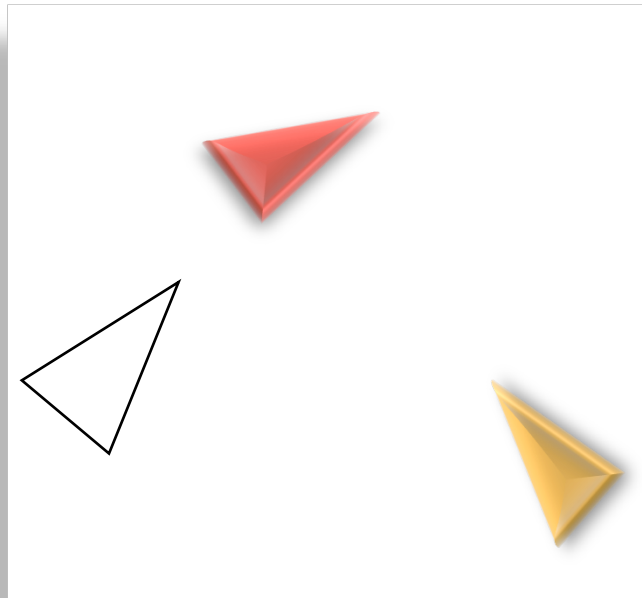
# Total Consistency (Alternating Execute)



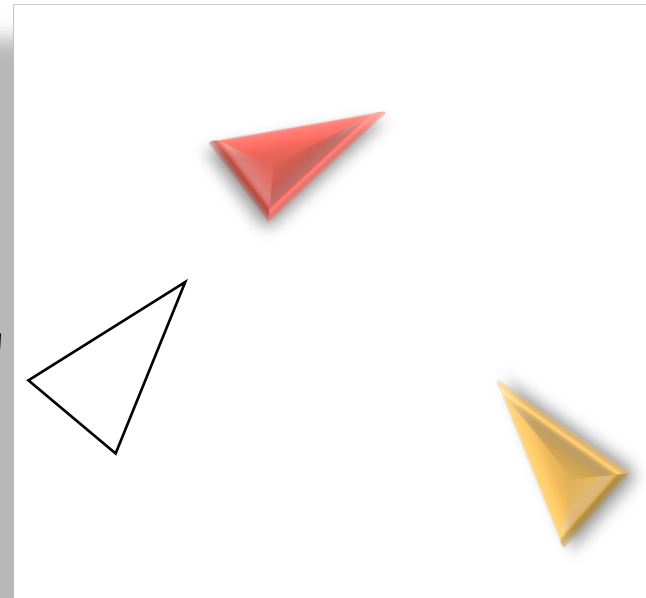
$$T = t + 50 \text{ ms} + 100 \text{ ms}$$

Delta T (latency) is 100ms

# Total Consistency (Alternating Execute)



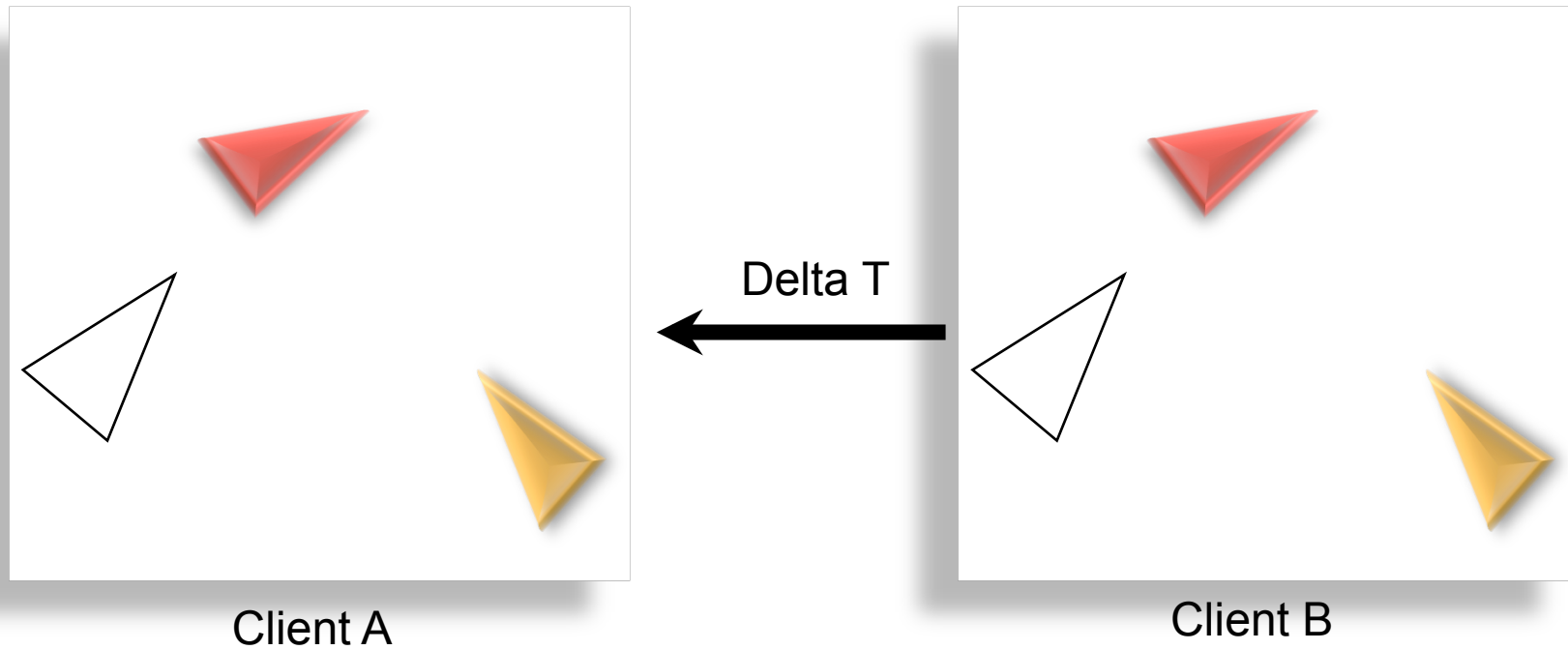
Client A



Client B

$$T = t + 50\text{ms} + 100\text{ms} + 50\text{ms}$$

# Total Consistency (Alternating Execute)



$$T = t + 50\text{ms} + 100\text{ms} + 50\text{ms} + 100\text{ms}$$

$$T = t + 300\text{ms}$$

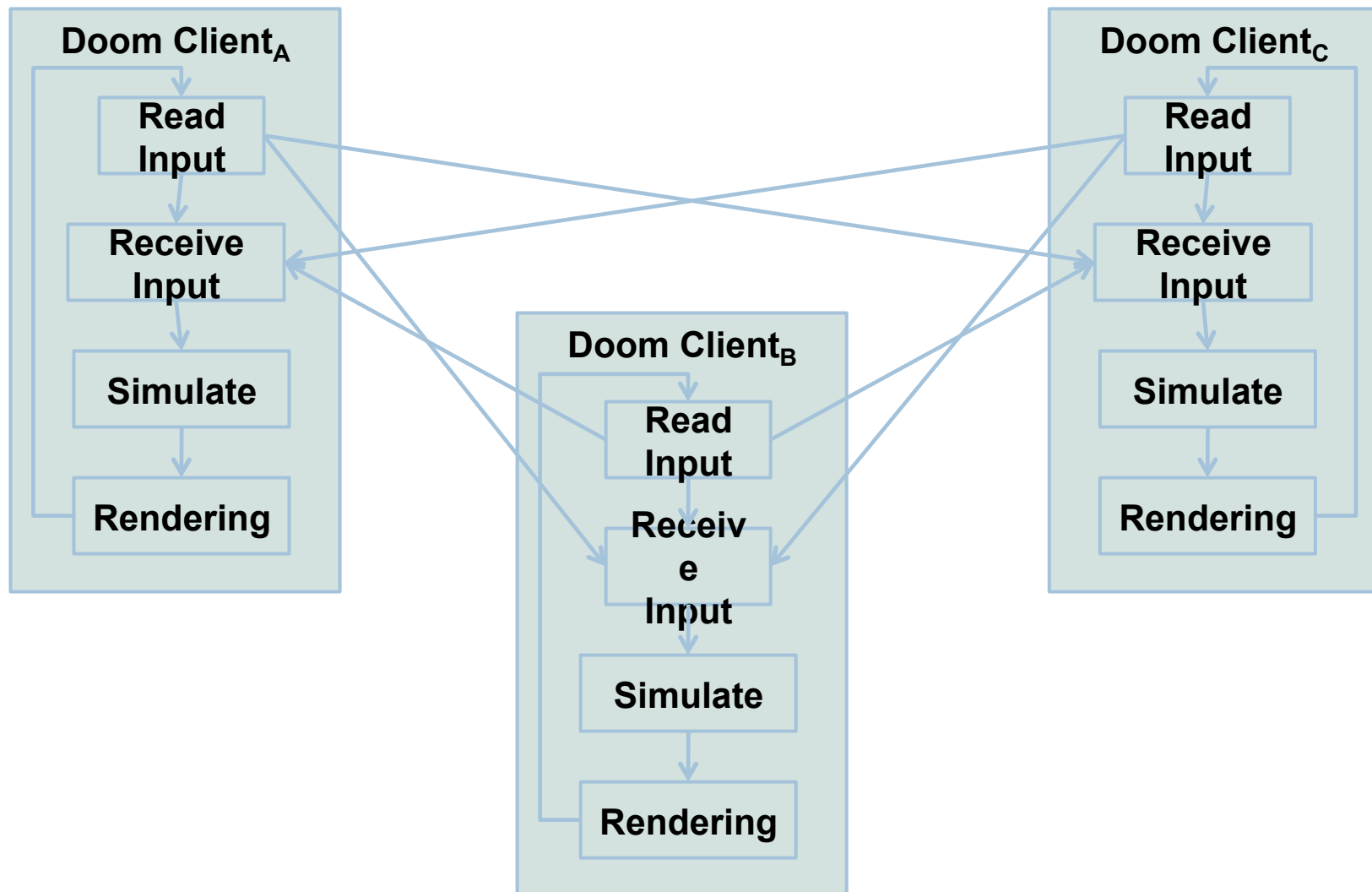
After 300ms Client A may move again!!!

# 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

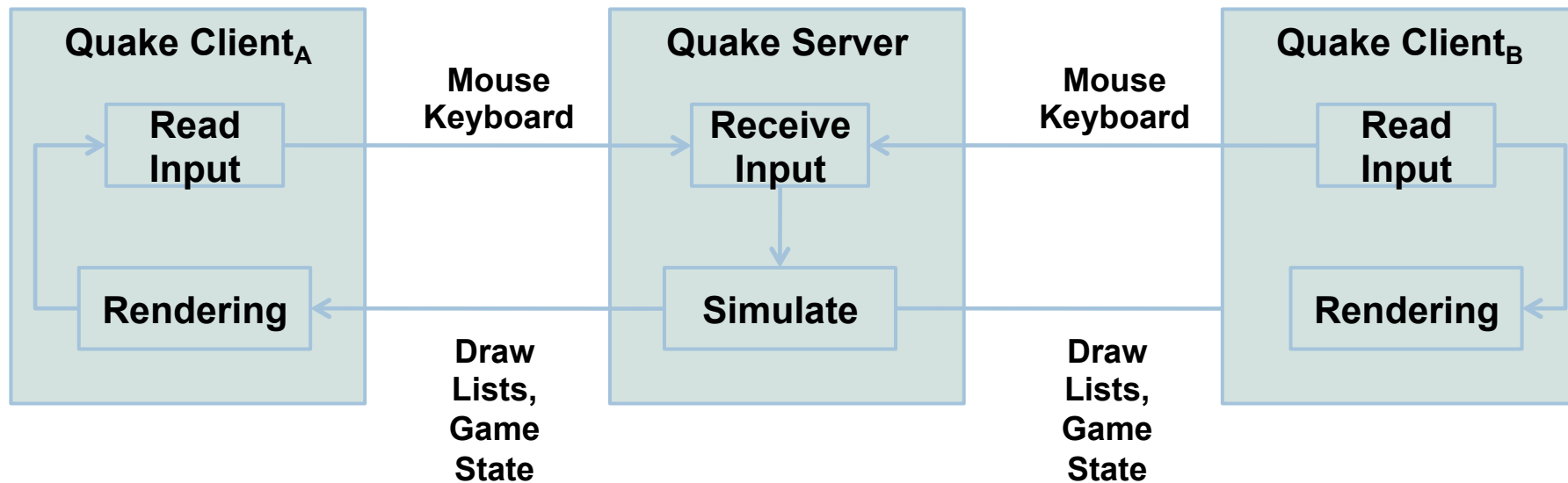


## Lock-Step (2)



- 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) – iD Software





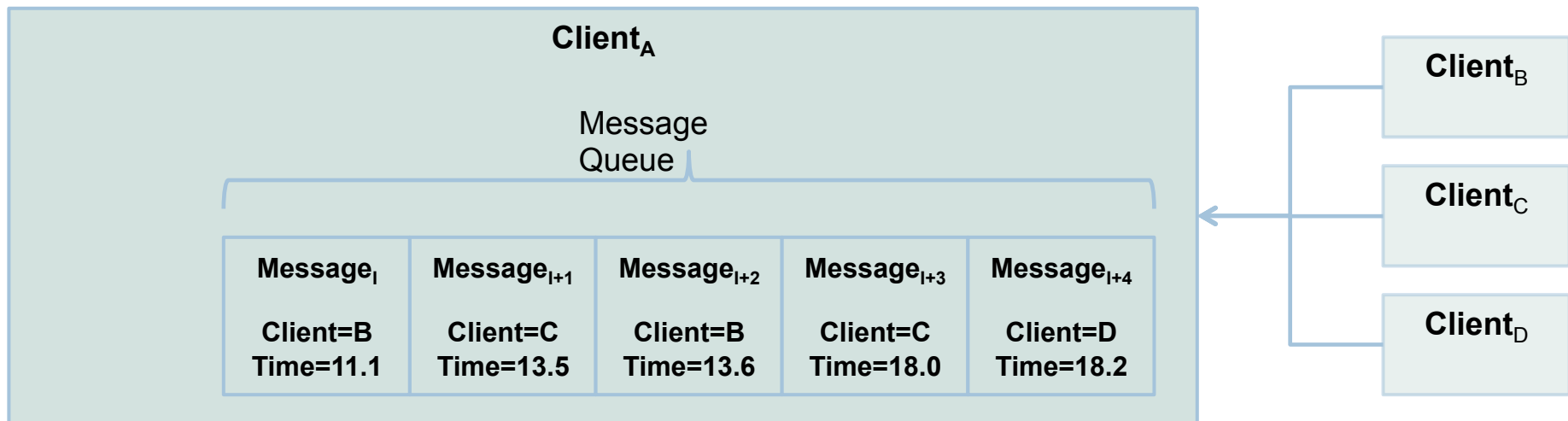
# CONSERVATIVE SIMULATIONS



# Conservative Simulations



- Lock-step are simple examples of conservative simulations
- Usually, there is no side-effect of the event you were waiting for
- E.G. in Quake, a lot of the time the other player's position is not important
  - ▣ Why wait for events? Why not just proceed
  - ▣ Answer is that you diverge IF you got shot
- However, for many simulations you can decouple event sequences



In a conservative simulation, events can be played out, if the simulation can know that another event cannot precede the ones it wants to play out. **In this case the first three messages can be played out**, but the fourth and fifth cannot.

# Notes



- Sufficient for many simulations
- Also known as *pessimistic simulations*
- Lots of theory about this: deadlocking, Chandy/Misra/Bryant *lookahead null message* algorithm
- See: Fujimoto, R. (2000) *Parallel and Distributed Simulation Systems*



TIME

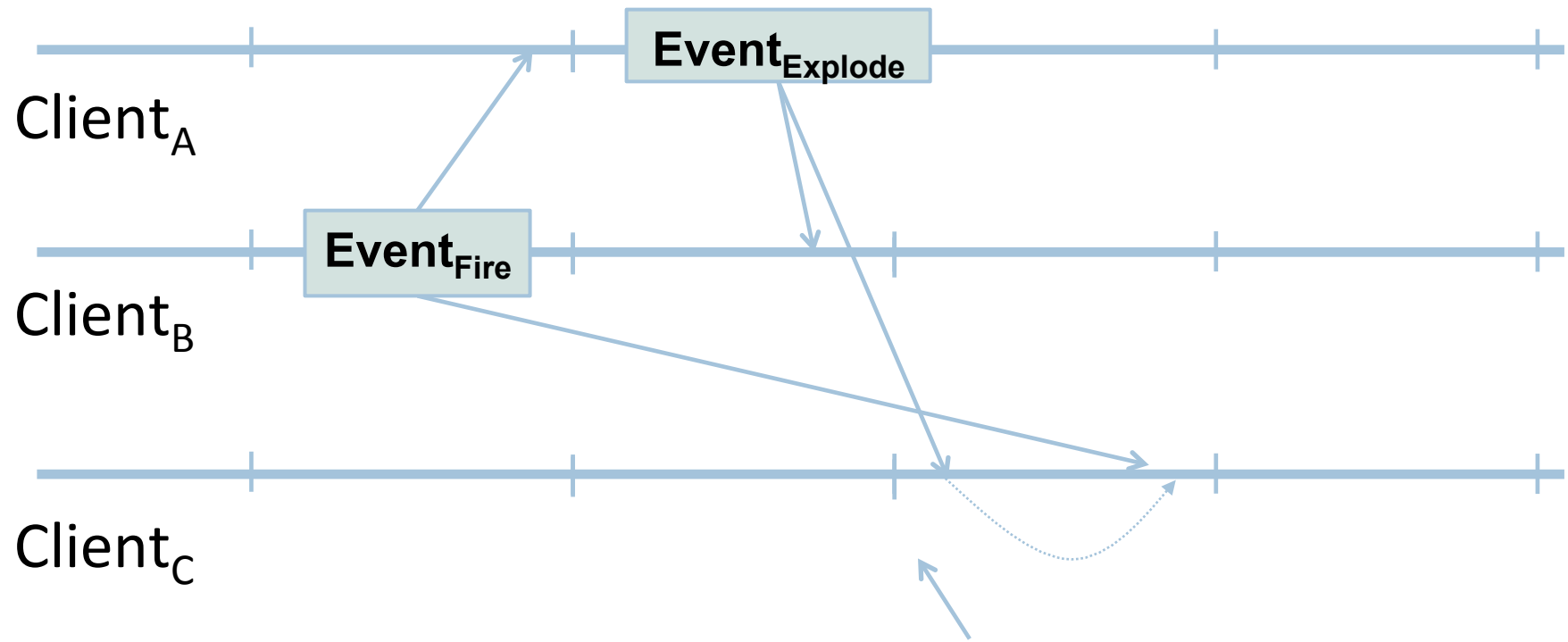
# Time



- Real-time synchronization needs a notion of time
- IF every event could be time stamped you could accurately reconstruct the recent past
- In reality clocks on machines can not be synchronized
- Can get close with *Network Time Protocol*
- Still not sufficient, applications tend to measure inter-client latency using round-trip times
-

# Virtual Time

- Sometimes it is sufficient to be able to order events
- Lamport's Virtual Time is really an event counter
- An event can indicate which events caused it, and which it depends on
- Thus, e.g. say  $\text{Event}_{\text{Explode}}$  caused  $\text{Event}_{\text{Fire}}$
- If  $\text{Event}_{\text{Explode}}$  says “ $\text{Event}_{\text{Fire}}$  caused me” then anyone who has  $\text{Event}_{\text{Explode}}$  waits for  $\text{Event}_{\text{Fire}}$
- This can be implemented for simple situations with just incremental counting ( $\text{Event}_{N+1}$  is held until  $\text{Event}_N$  is played)



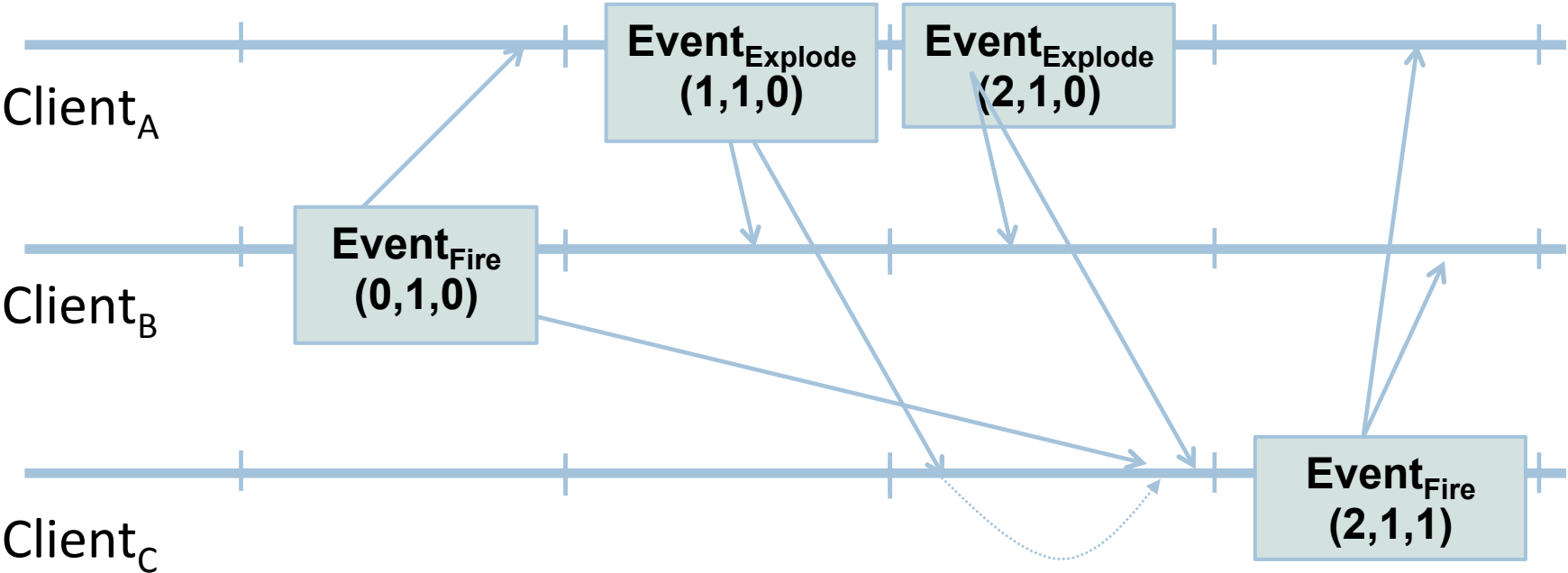
Event<sub>Explode</sub> is delay at Client<sub>C</sub> until after Event<sub>Fire</sub>  
 A causal ordering scheme prevents Client<sub>C</sub> from seeing an explosion before the fire event that caused it. In this case, the timeline and the ticks on the timeline only serve to indicate the passage of wall clock time, they don't indicate time steps.

# For Large Simulations



- Practically this can be achieved with vector clocks
- Each simulation keeps an event order of the events it received, and then states which events it had received when it generated an event





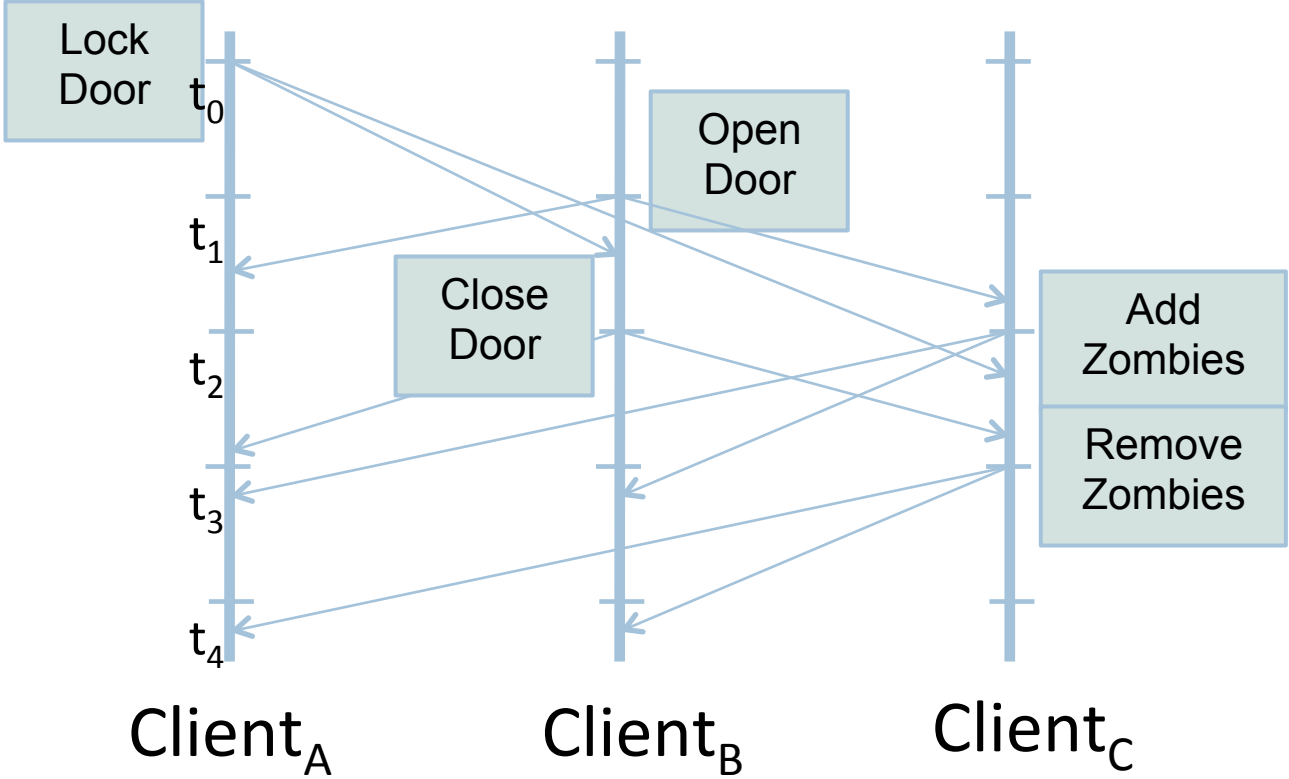


# OPTIMISTIC ALGORITHMS

# 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)



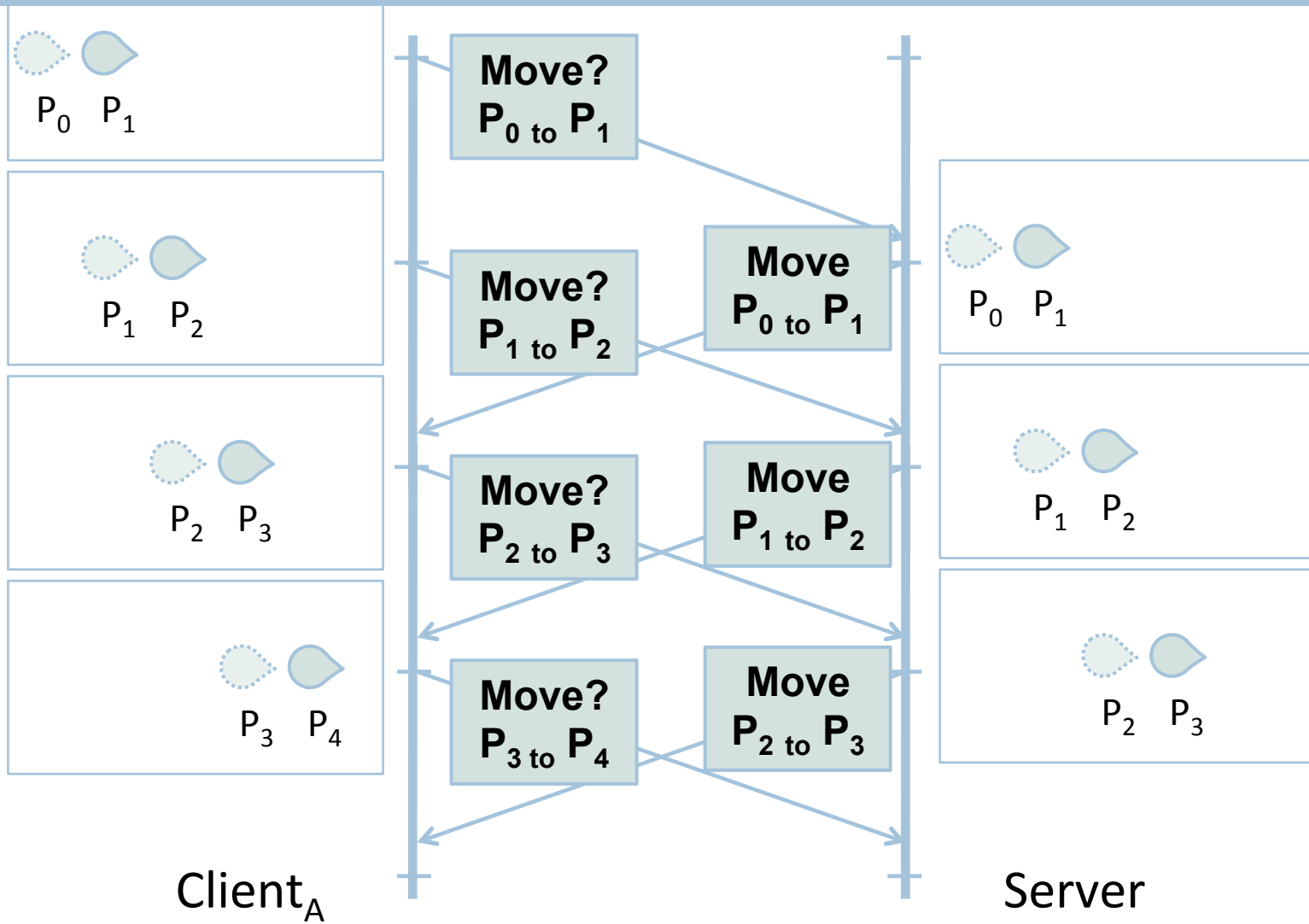


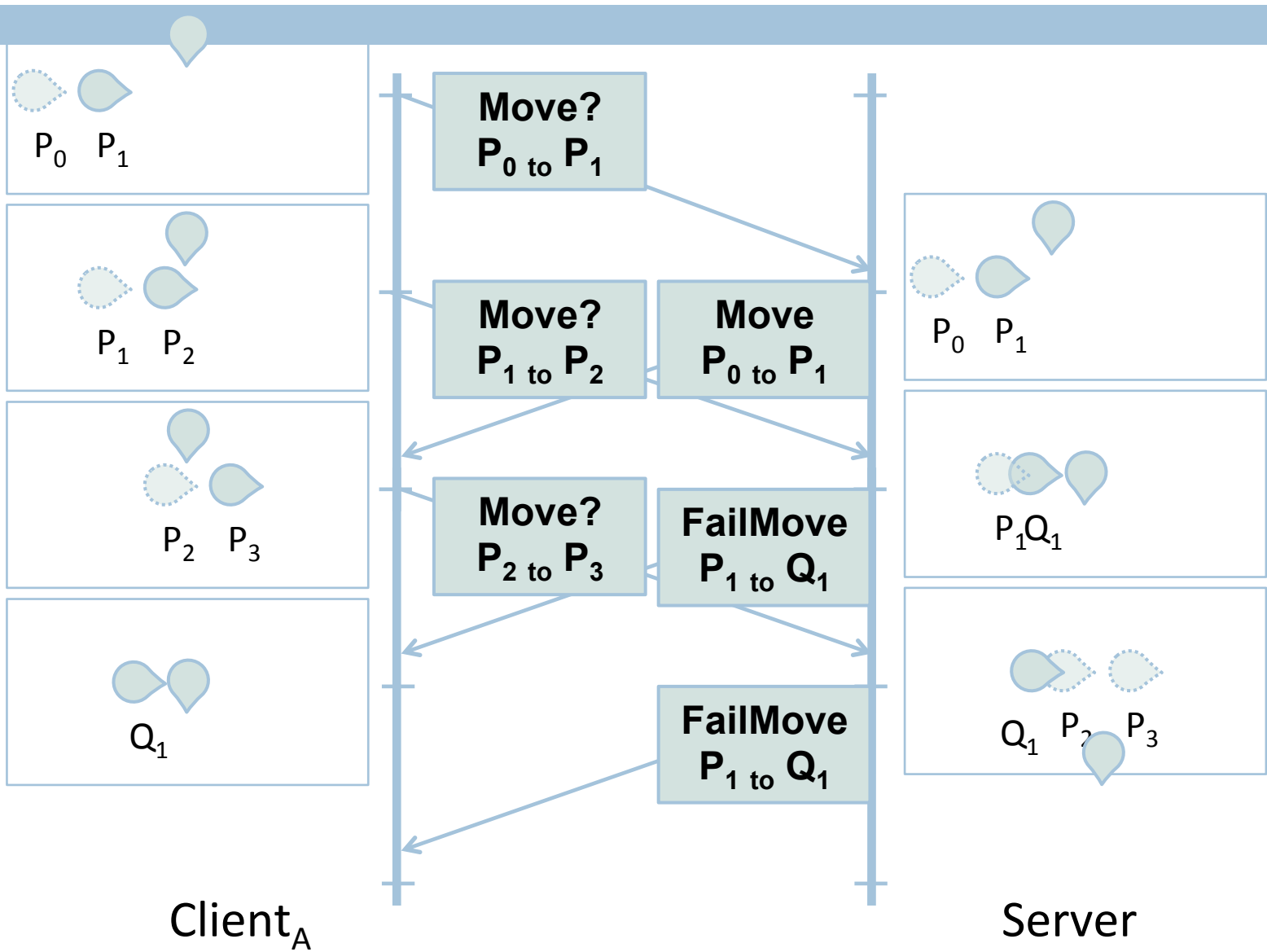
CLIENT PREDICT AHEAD

# Predict Ahead

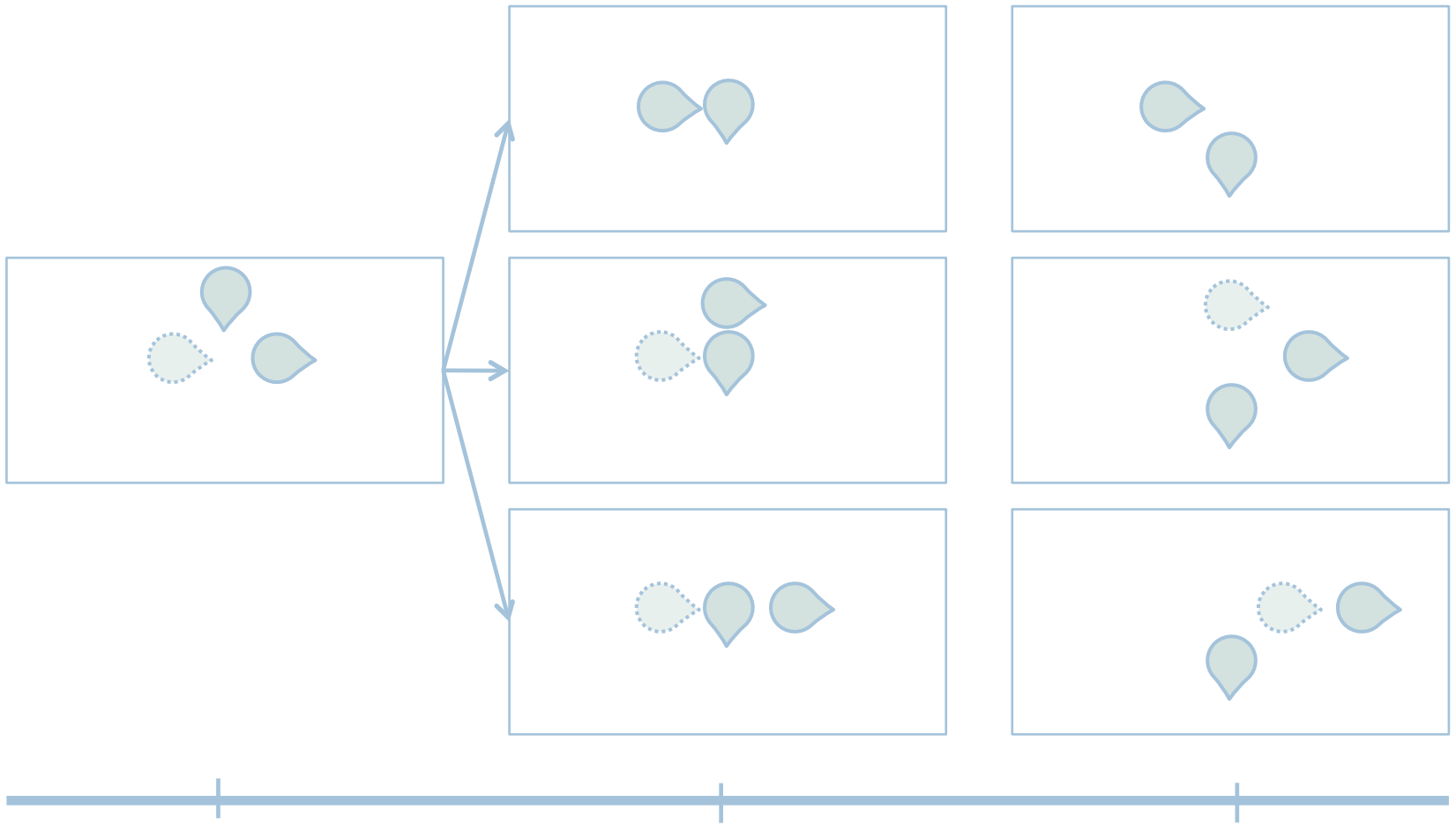


- A form of optimism: assume that you can predict what a server (or another peer) is going to do with your simulation
- Very commonly applied in games & simulations for your own player/vehicle movement
- You assume that your control input (e.g. move forward) is going to be accepted by the server
- If 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!)











# EXTRAPOLATION ALGORITHMS

# 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

- 1<sup>st</sup> order model

$$P_1 = P_0 + (t_1 - t_0) \cdot V_0$$

- 2<sup>nd</sup> order model

$$V_1 = V_0 + (t_1 - t_0) \cdot A_0$$

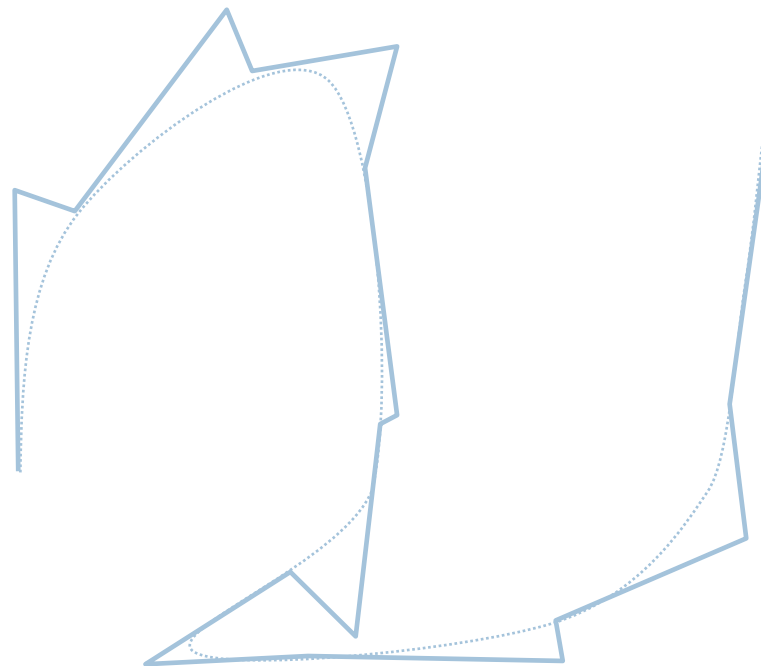
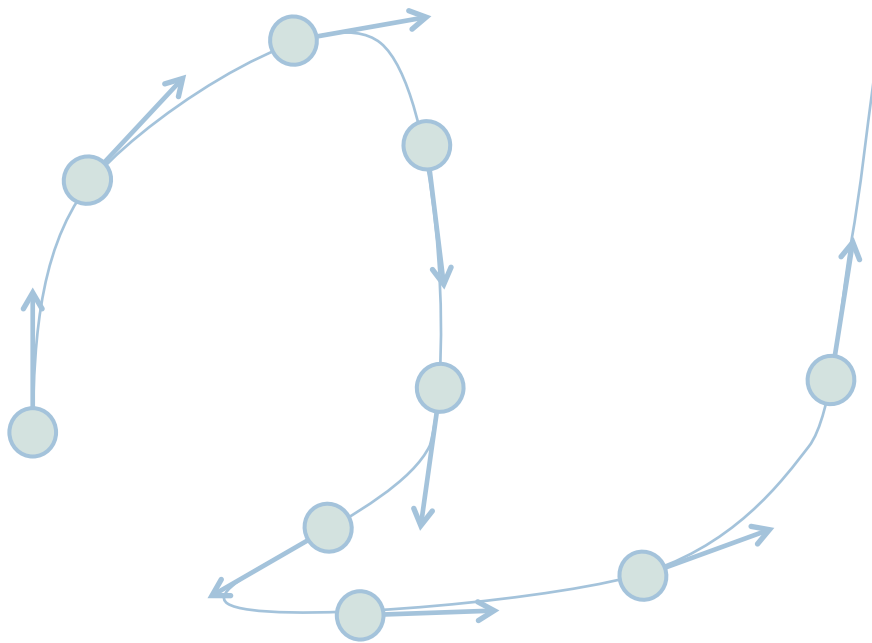
$$P_1 = P_0 + (t_1 - t_0) \cdot V_0 + \frac{1}{2} \cdot A_0 \cdot (t_1 - t_0)^2$$

# When to Send Updates

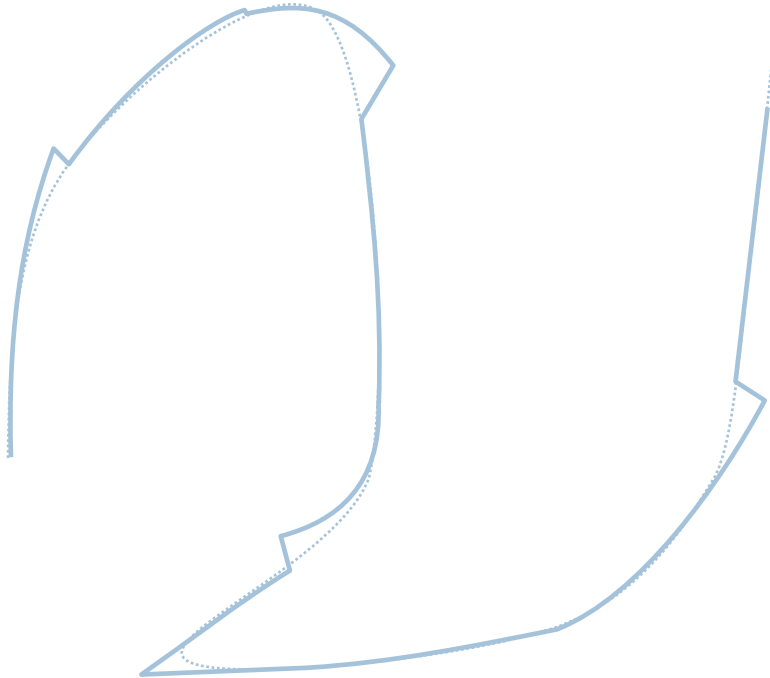
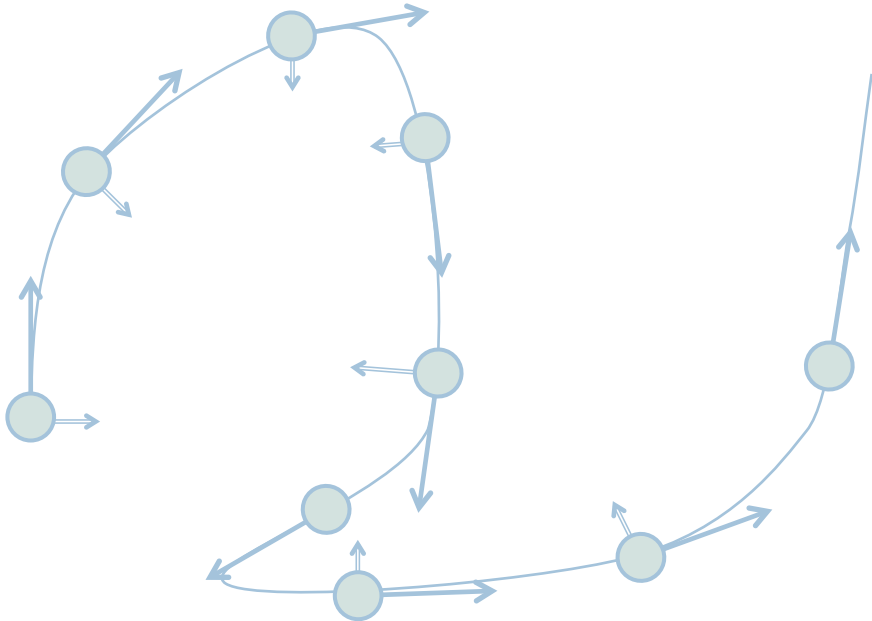


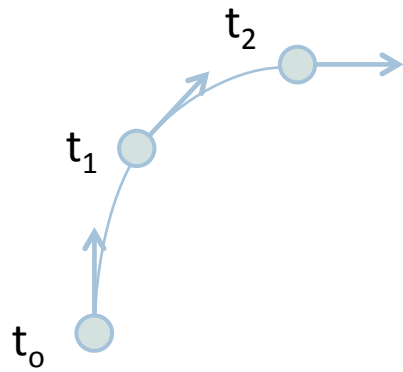
- Note that if this extrapolation is true you never need to send another event!
- It will be wrong (diverge) if acceleration changes
- BUT you can wait until it diverges a little bit before sending events
- The sender can calculate the results as if others were interpolating (a ghost), and send an update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

# 1<sup>st</sup> Order Model

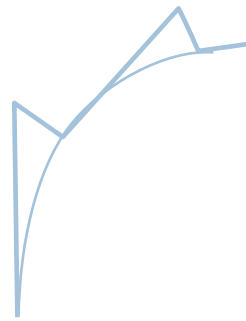


2<sup>nd</sup> Order Model

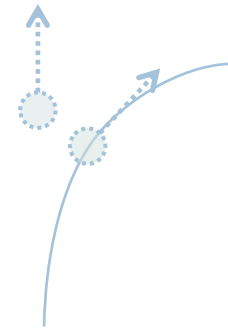




a) Player model sending three updates



b) Ghost model path without blending



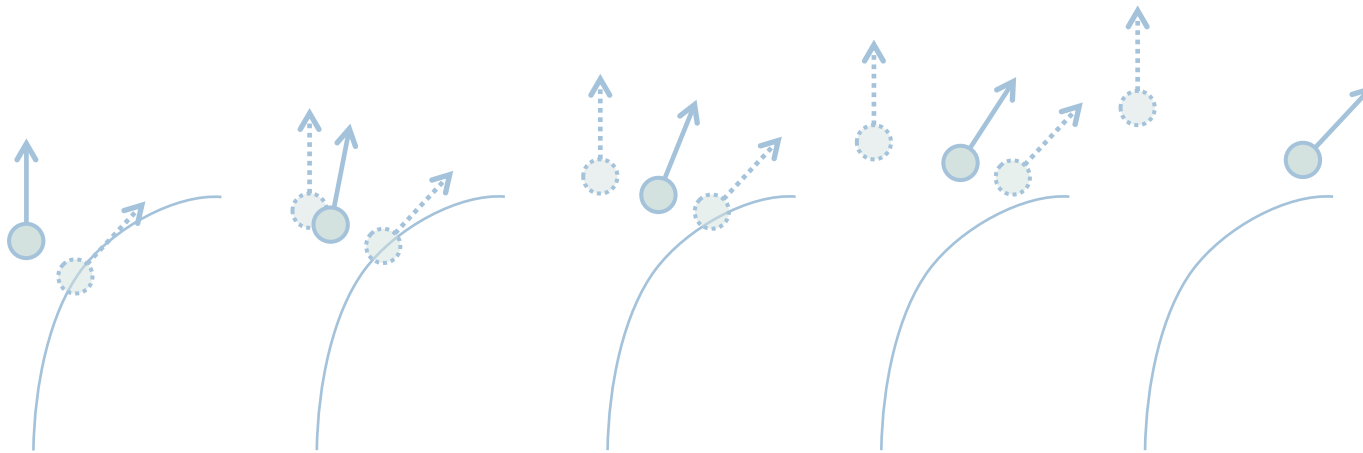
c) Old ghost model and new ghost model at  $t_1$



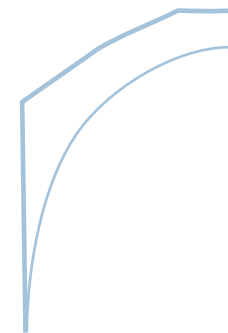
# Convergence Algorithm



- 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. would mean slipping)



d) Blending between the old ghost and new ghost over several frames

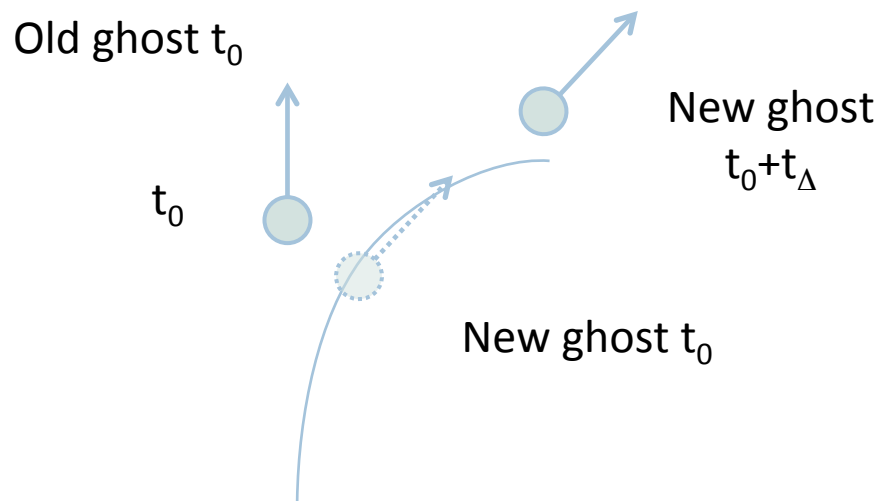


e) Ghost model path with blending

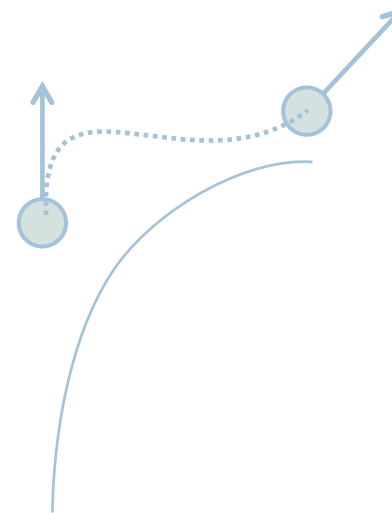
# Convergence Algorithm



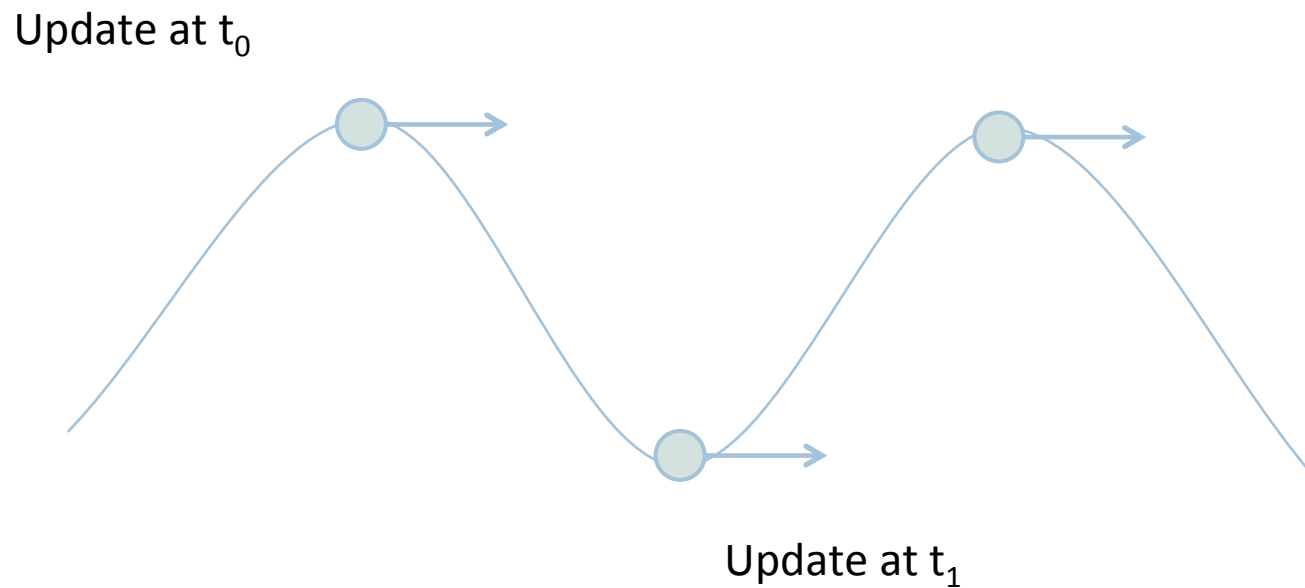
- So you could steer the vehicle to correct its own position
  - ▣ This has frequency instabilities
  - ▣ Deals badly with obstacles as the interpolated path isn't the same as the real path



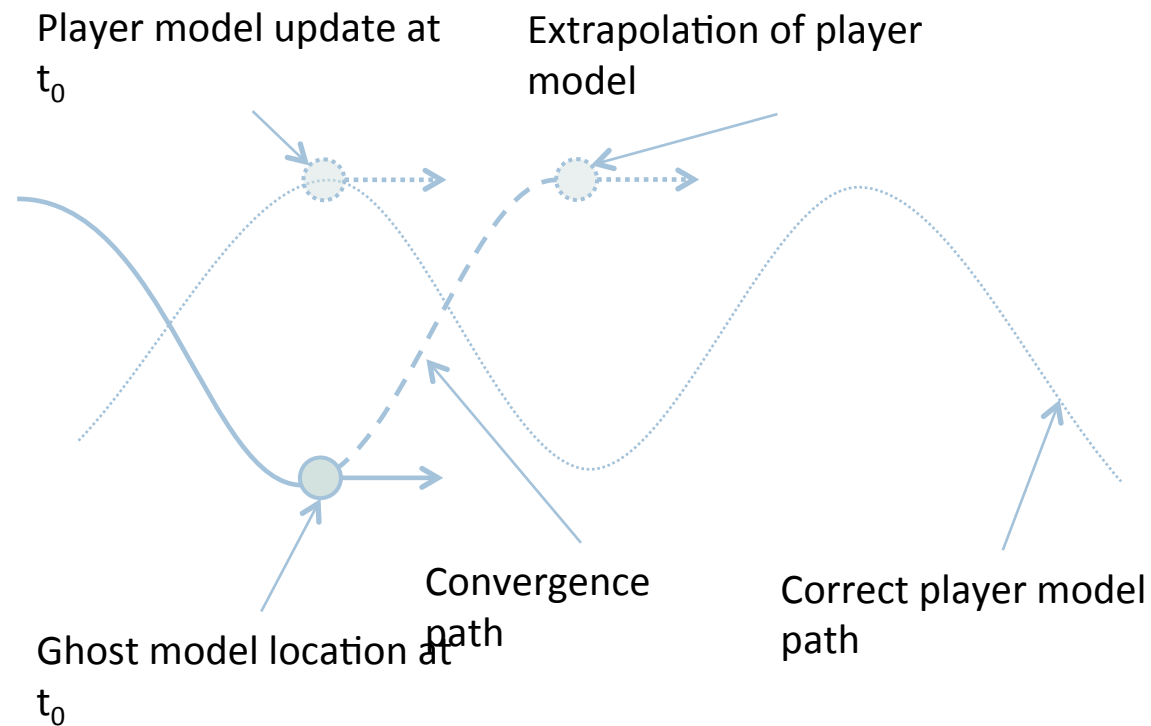
a) Old ghost position at  $t_0$ , new ghost position at  $t_0$  and new ghost position at  $t_0+t_\Delta$



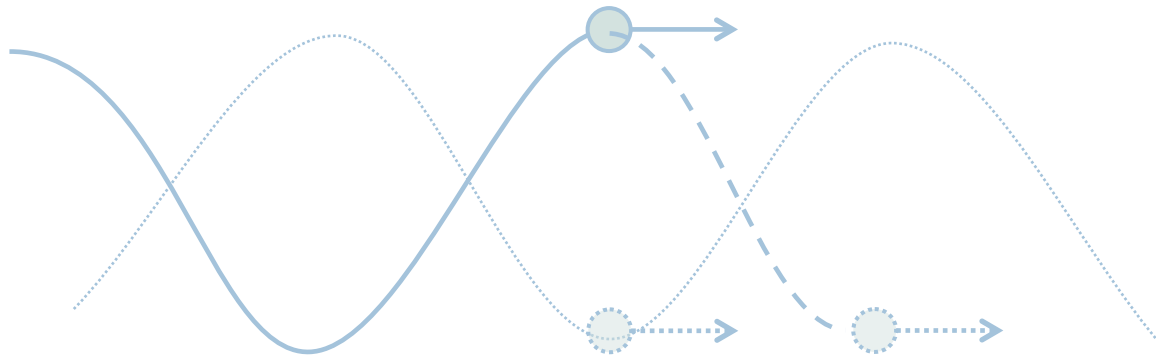
b) Dotted line shows the planned path to reach the target position and direction



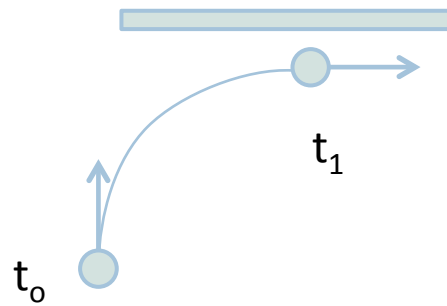
a) Player model showing the timings of dead-reckoning updates at the peaks of a periodic motion



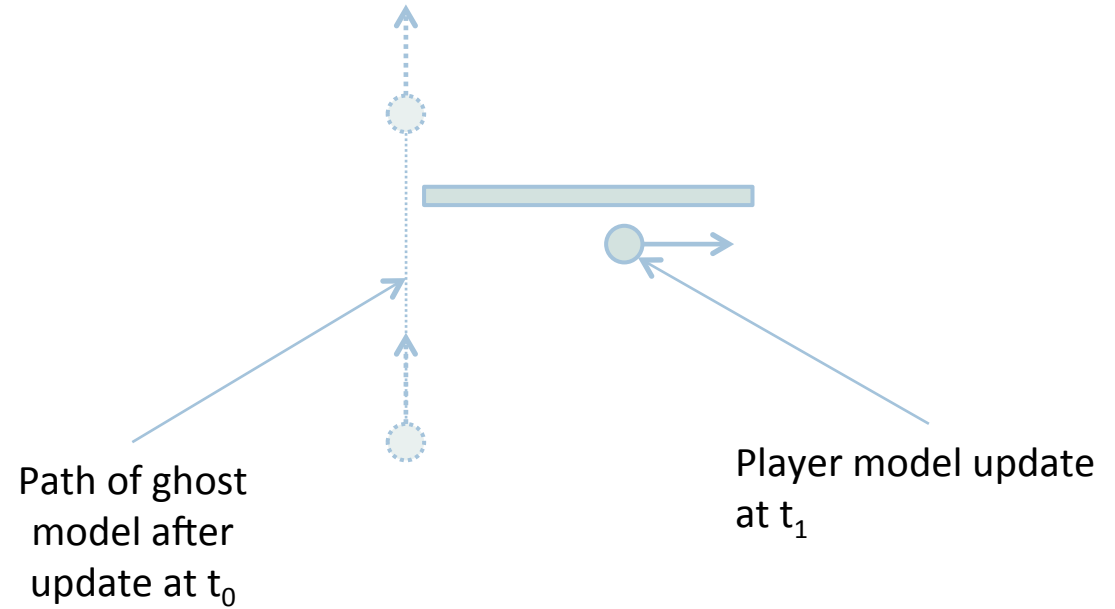
- b) On arrival of an update message, the ghost model plans to converge the current ghost model position with an extrapolation of the received position



c) On the next update message the ghost model is out of phase with the player model. T



a) Player model showing the object avoiding the wall



b) After the update at  $t_1$  the ghost model cannot converge



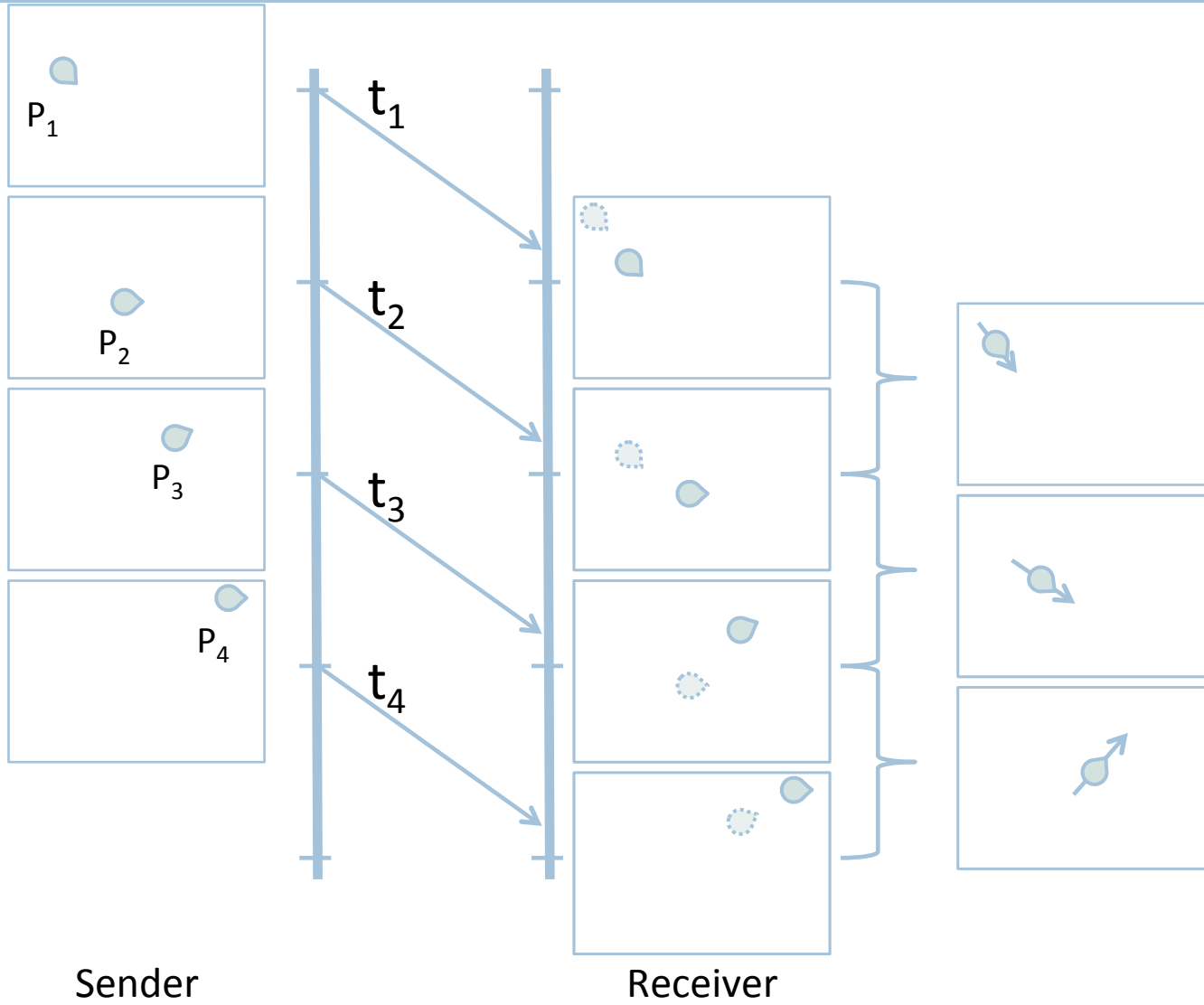


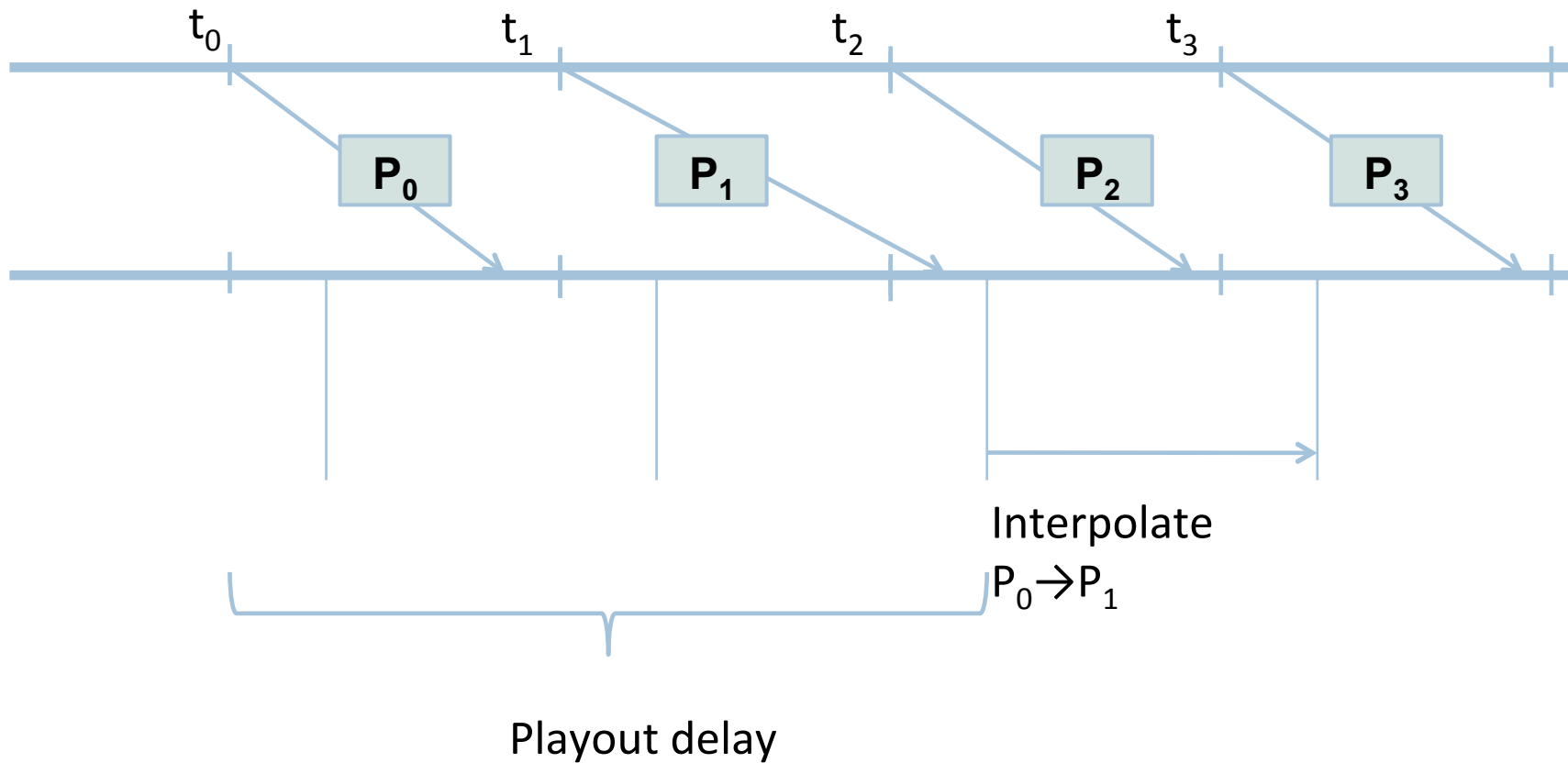
# INTERPOLATION, PLYOUT DELAYS AND LOCAL LAG

# 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

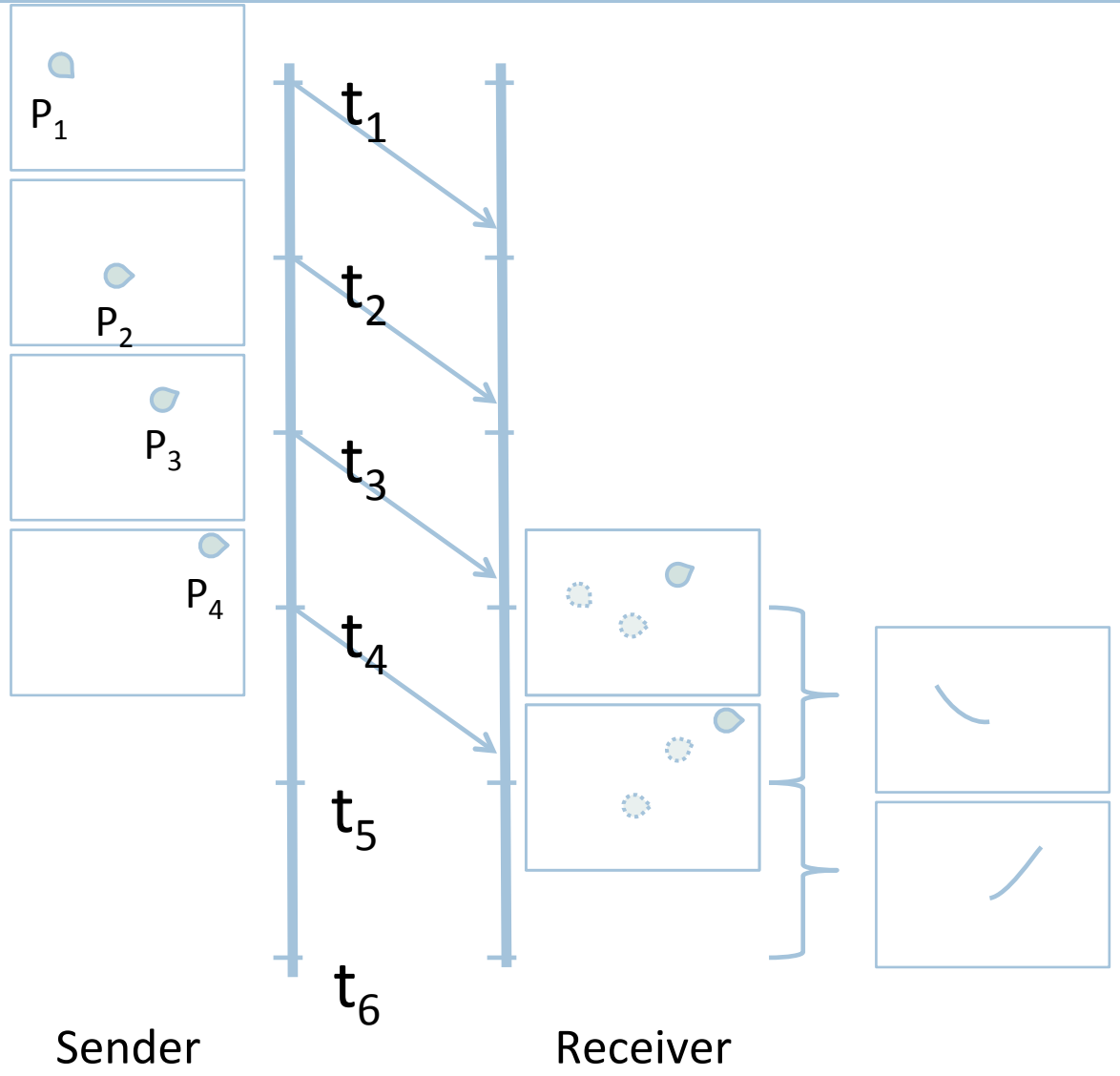




# Non-Linear Interpolation



- Need to consider several aspects
- Object movement is not linear, so could use quadric, cubic, etc. by keeping three or more updates

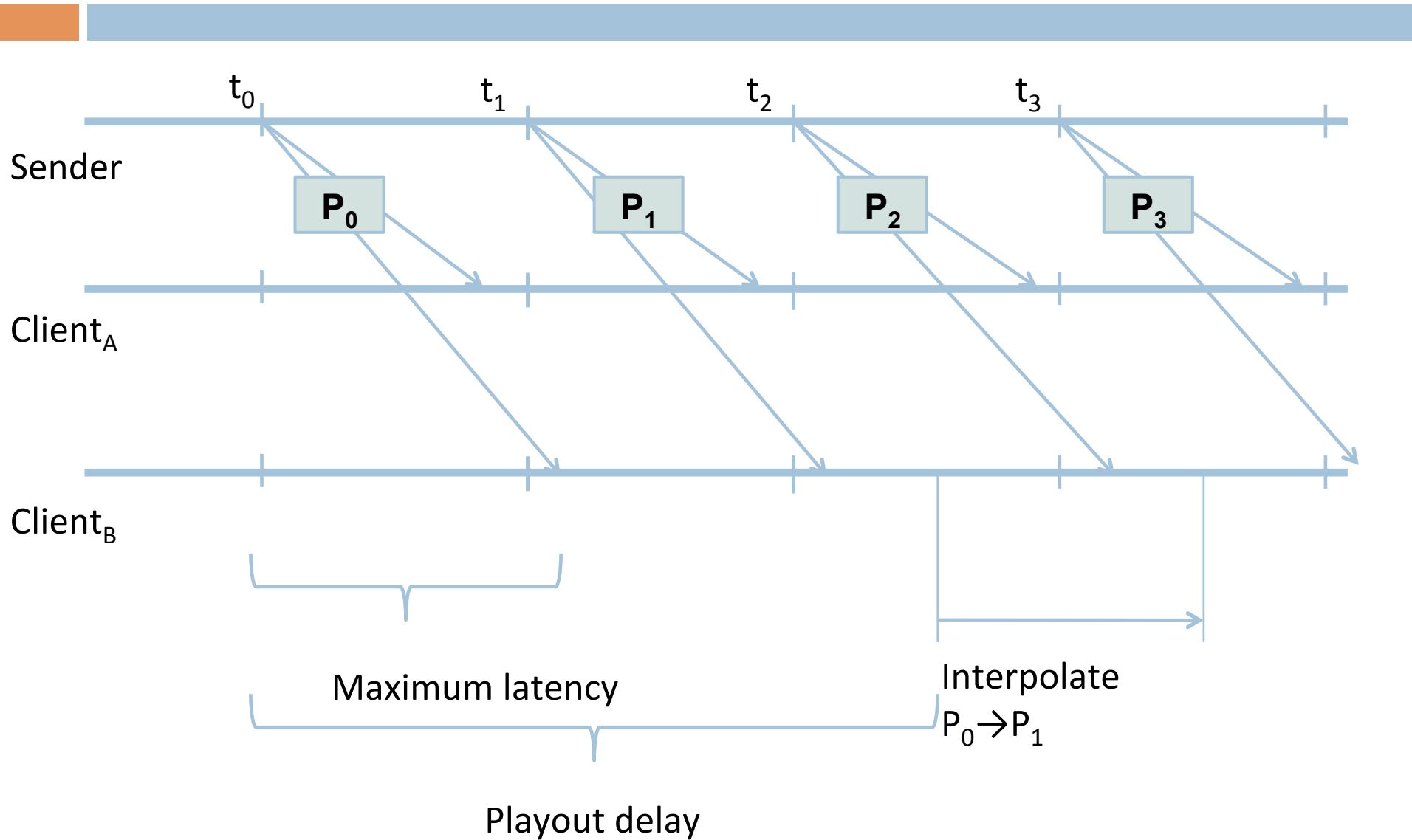


# Playout Delays



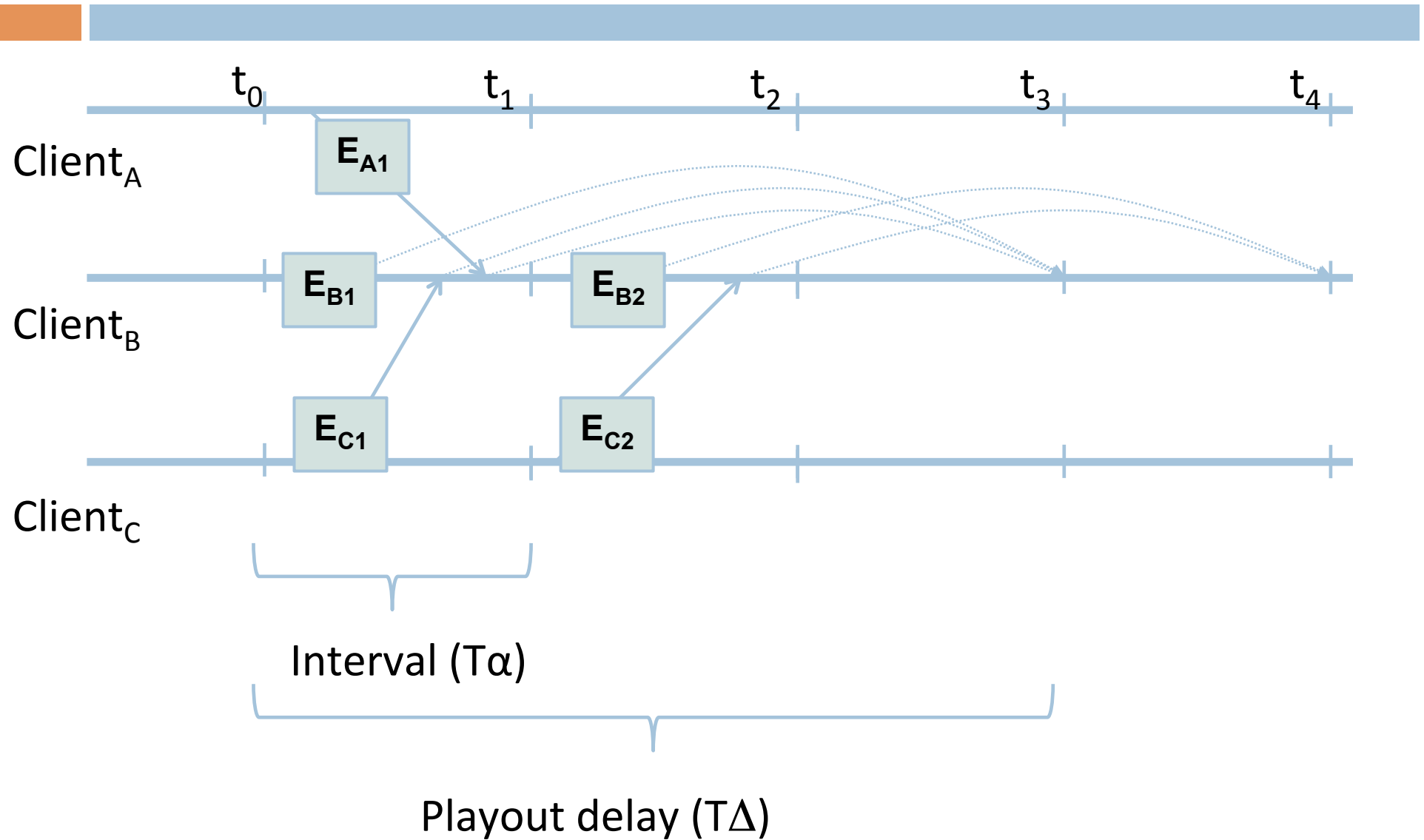
- Note that jitter is not uniform, you need to be conservative about how long to wait (if a packet is late you have no more information to interpolate, so the object freezes)
- NVEs and NGs thus sometimes use a *playout delay*
- Note that if you use a playout delay on the clients own input, then all clients will see roughly the same thing at the same time!
- A strongly related technique is *bucket synchronisation*, pioneered in the seminal MiMaze

# Playout Delay





# Bucket Synchronization





# PERCEPTION FILTERS

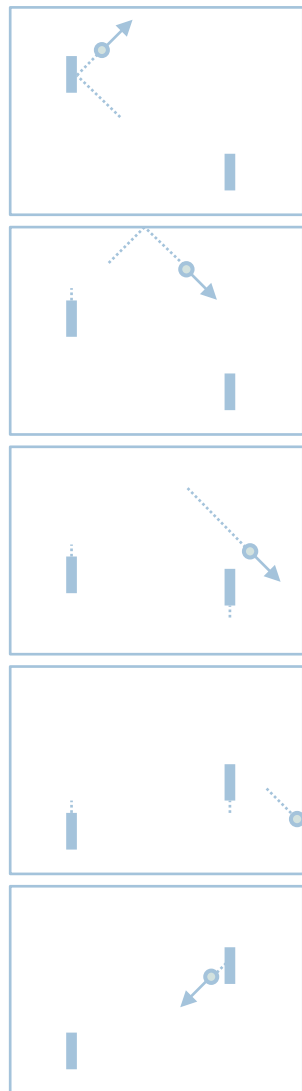
# Perception Filters



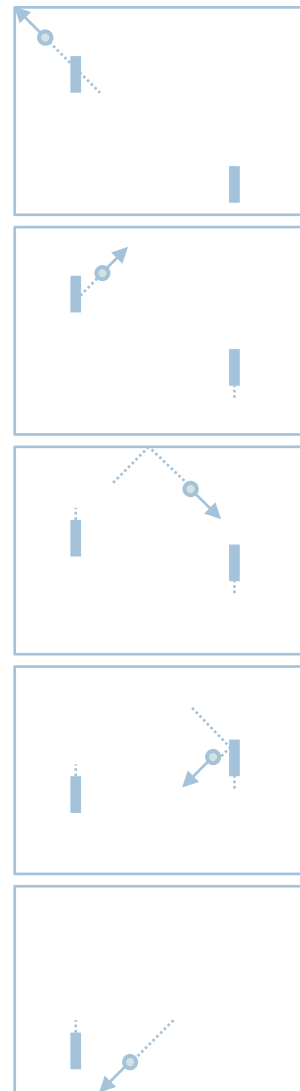
- In these techniques, the progress of time is altered at different clients
- Clients choose to predict ahead or delay playout depending on the meaning and their expected interaction



Client<sub>A</sub>



Client<sub>B</sub>

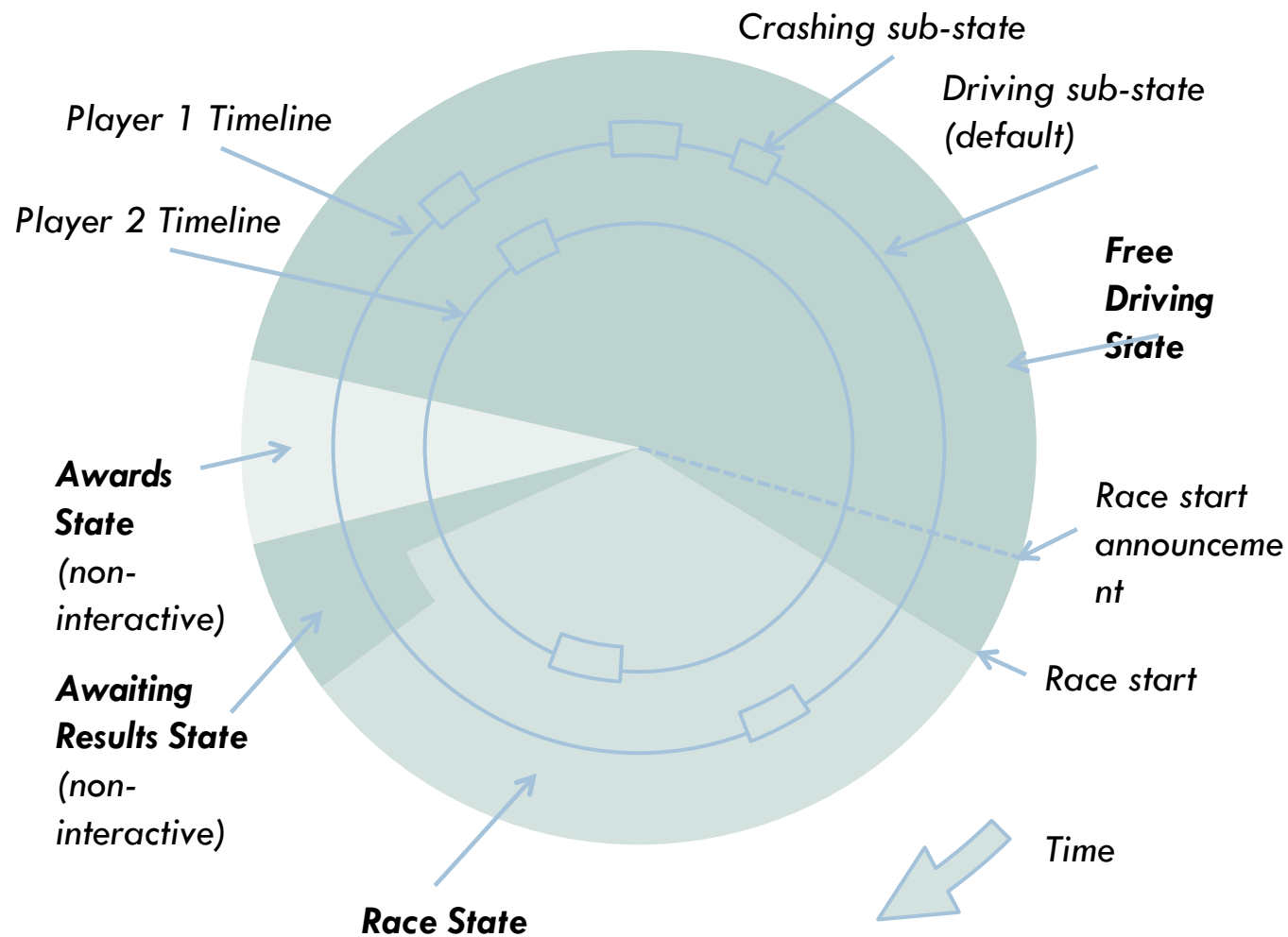




# CASE STUDY: BURNOUT <sup>TM</sup> PARADISE

# Burnout™ Paradise





Scalability

# Introduction to Networked Graphics

IEEE Virtual Reality 2011







- *Scalability*

- - Management of awareness

- - Interest specification

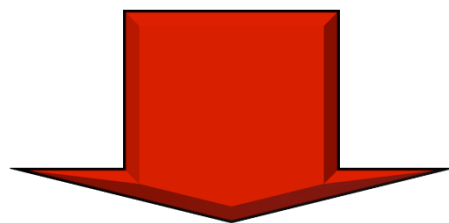
- - Server partitioning



# GOALS FOR SCALABILITY

# Interest Specification

- A user is NOT an omniscient being
- A user is NOT interested in every event
- A client is NOT able to process everything



- **Just give each client enough to instil the user's illusion of an alternate reality**
  - **Interest: visibility, awareness, functional, ...**
  - **Network and computational constraints**

# 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 or nearby
  - ▣ 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





# SPATIAL PARTITIONING

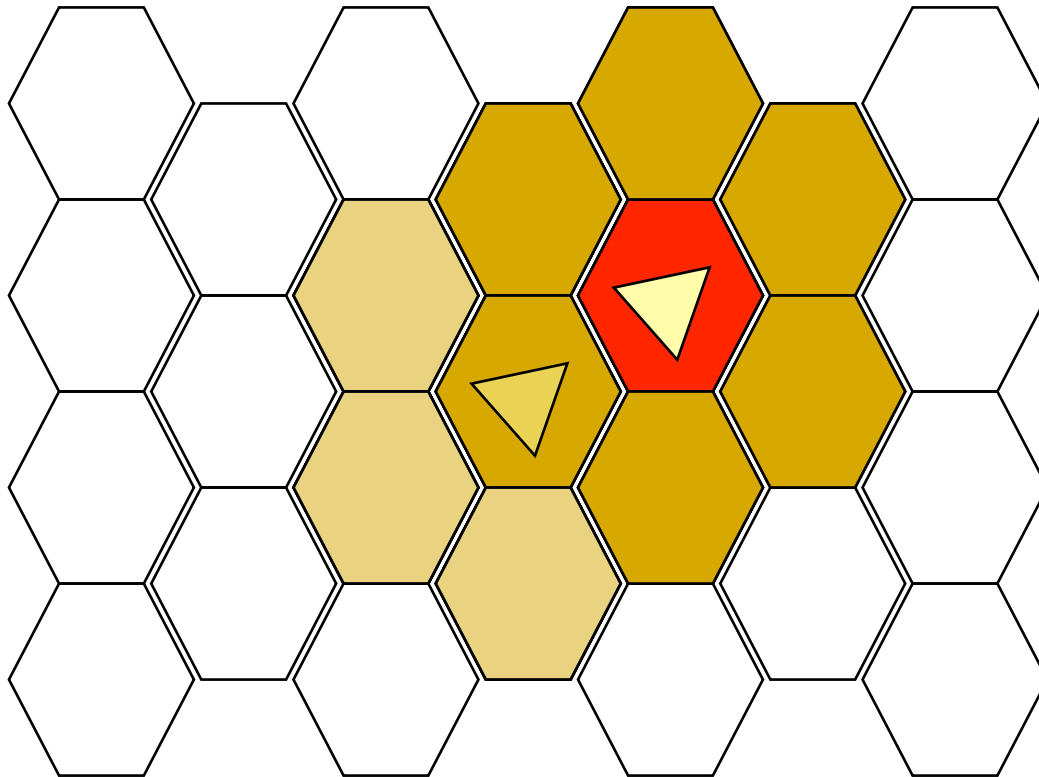


# Spatial Partitions



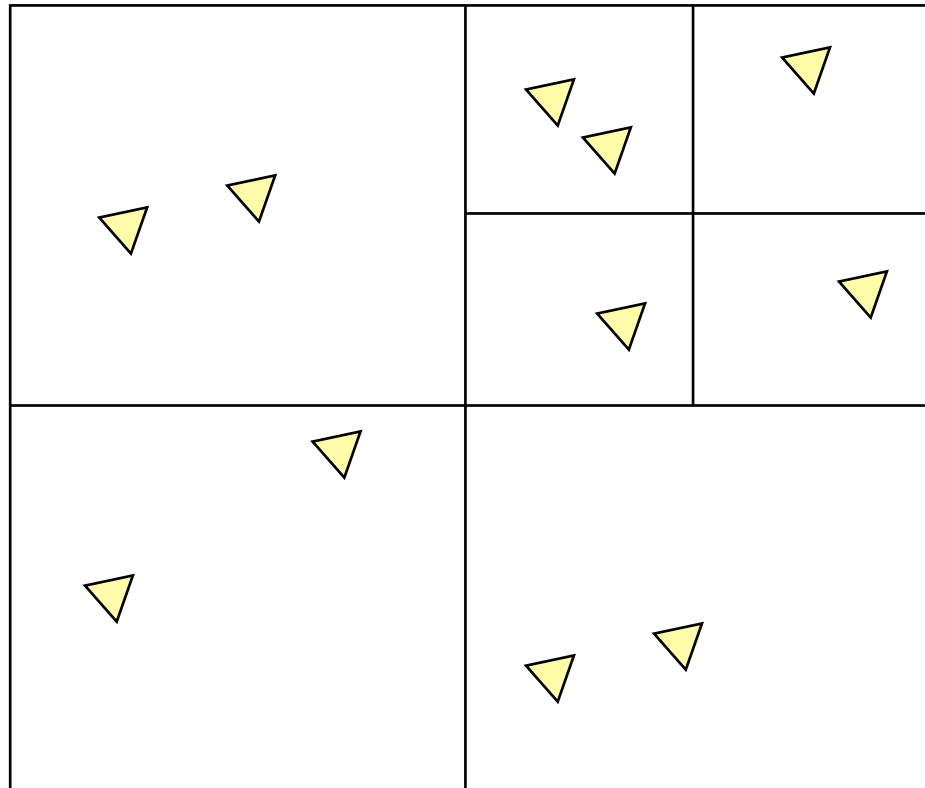
- Global Partitions
  - Static Grid
  - Hierarchical Grid
  - Locales
- Local Partitions
  - Aura
  - Visibility
  - Nearest Neighbours

# Global Partitions: Static Cells



- 1 Cell = 1 Group
- Hexagon regular shape
- Tied into the grid – static
- Send current cell
- Receive neighbours
- Any architecture (distributed)

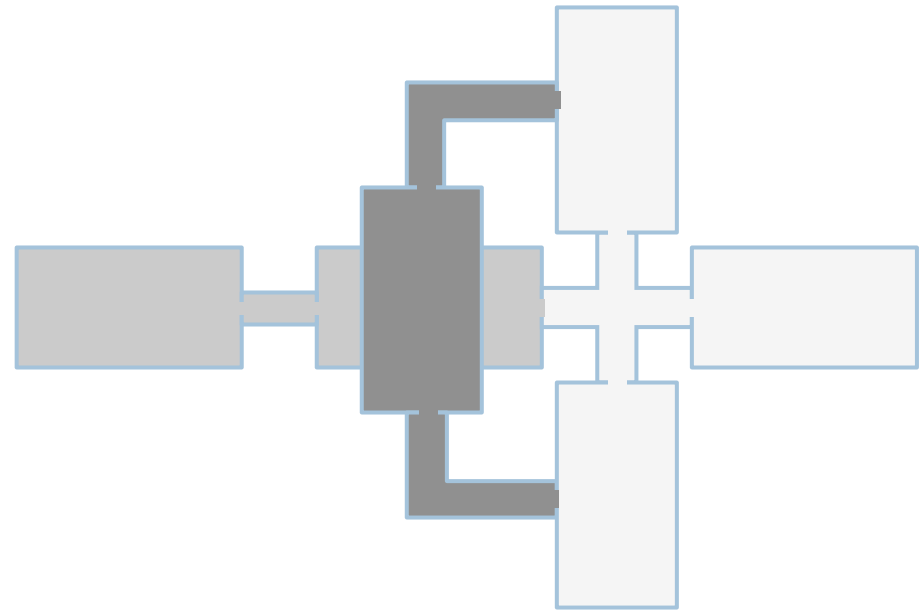
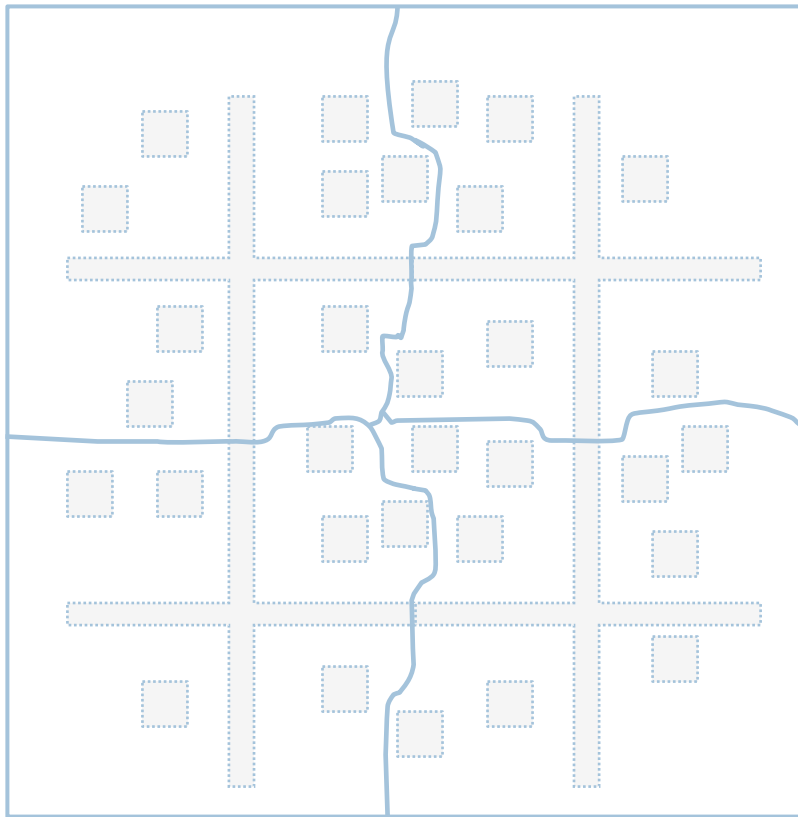
# Global Partitions: Hierarchical Grid



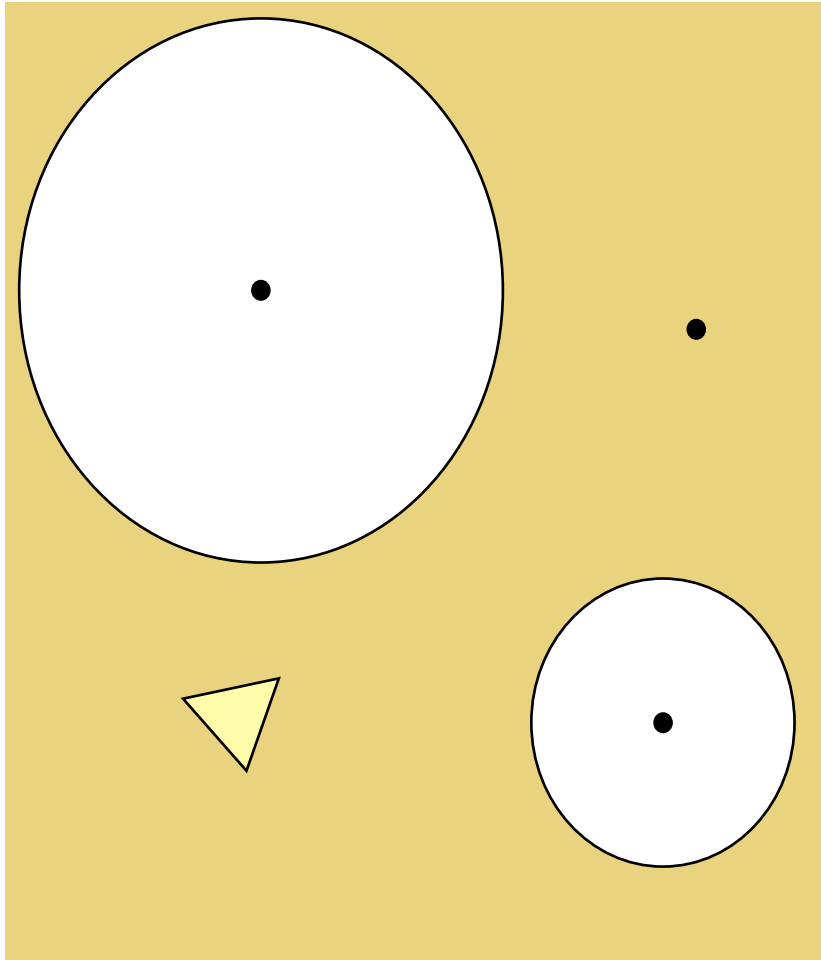
Threshold = 5

- 1 Cell = 1 Group
- Square cells
- Send current cell
- Receive current cell
- Any architecture (distributed)
- Exceeds threshold, expand

# Global Partitions: Irregular

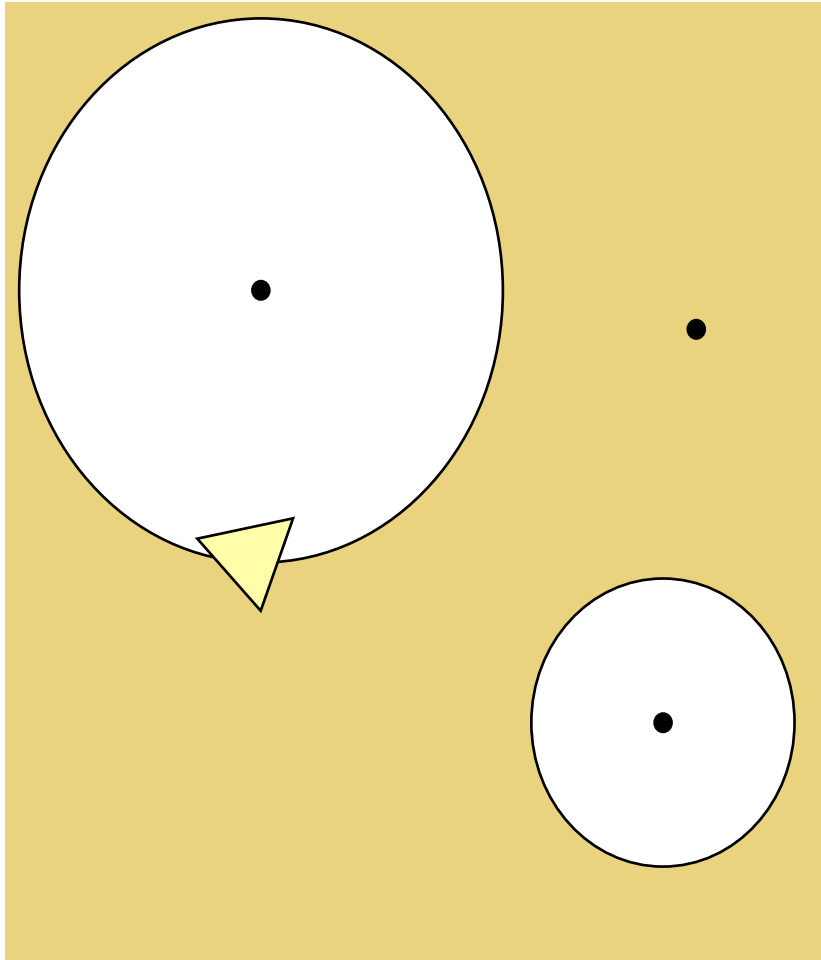


# Global Partitions: Locales



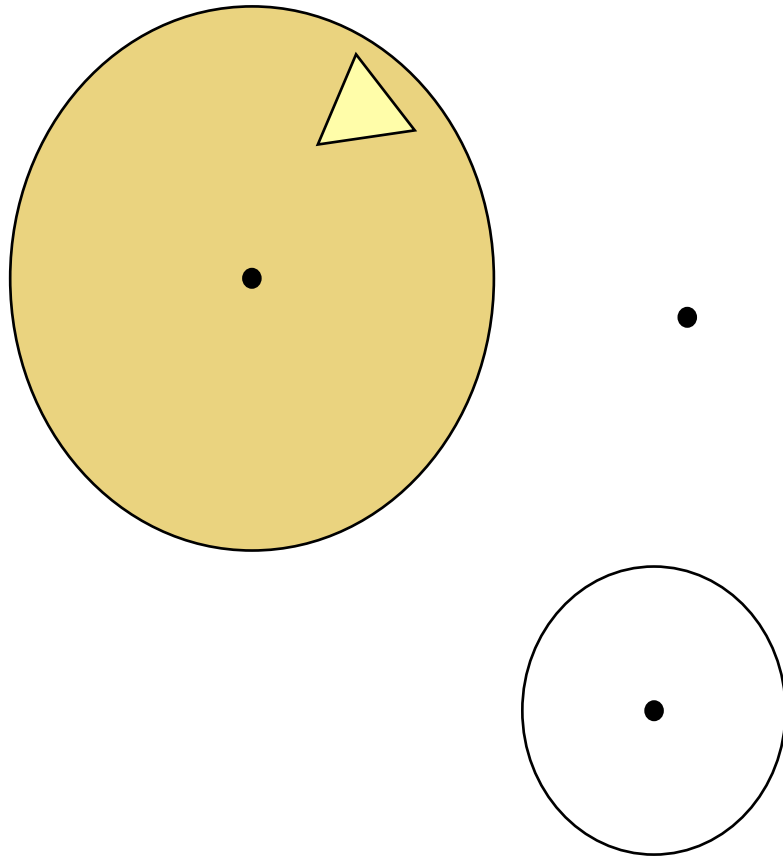
- 1 locale = 1 group
- Locale is arbitrary shape
- Locale placement is static
- Associated transform matrix
- Any architecture (distributed)

# Global Partitions: Locales



- 1 locale = 1 group
- Locale is arbitrary shape
- Locale placement is static
- Associated transform matrix
- Any architecture (distributed)

# Global Partitions: Locales



- 1 locale = 1 group
- Locale is arbitrary shape
- Locale placement is static
- Associated transform matrix
- Any architecture (distributed)

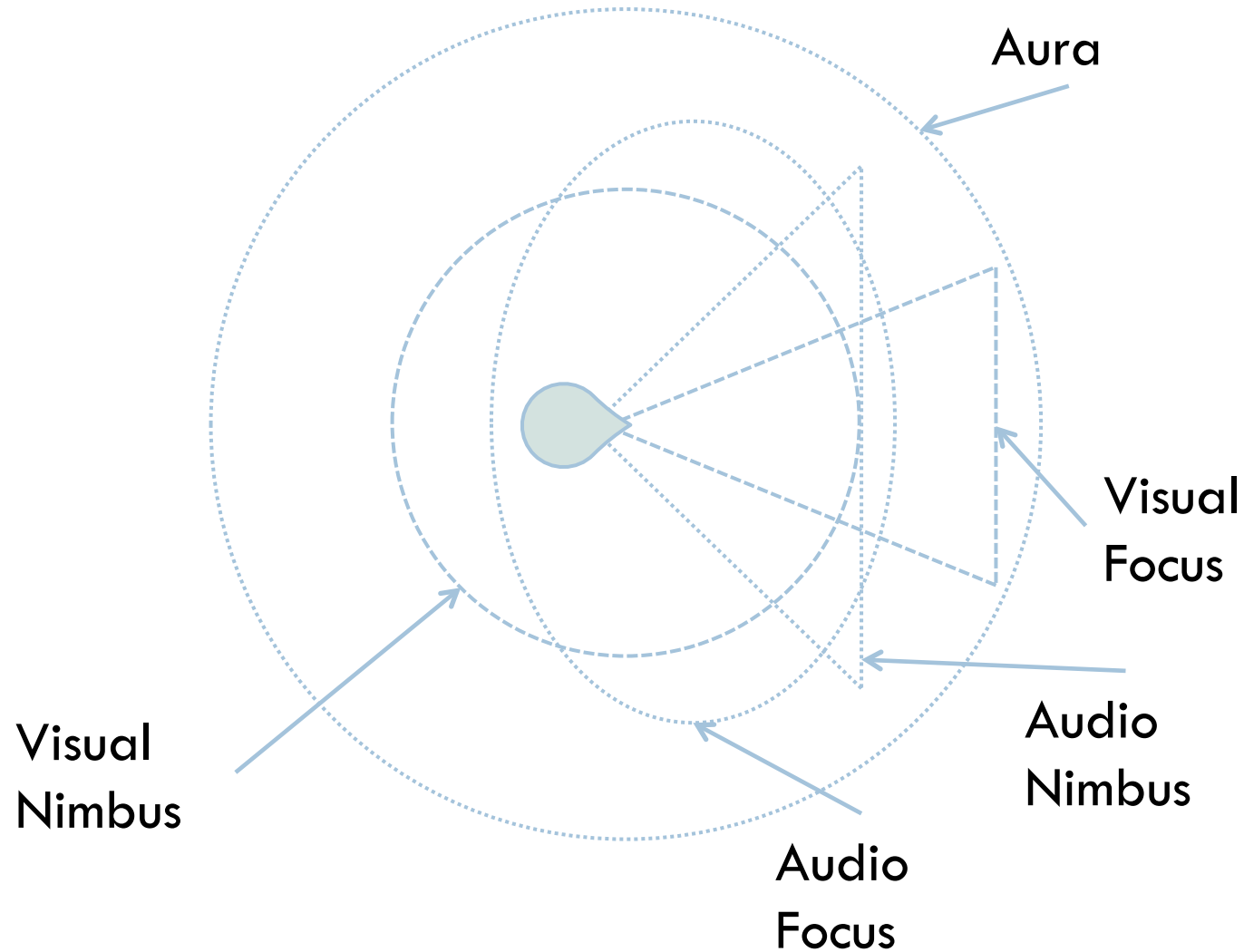
# Local Partitions: Aura, Focus, Nimbus



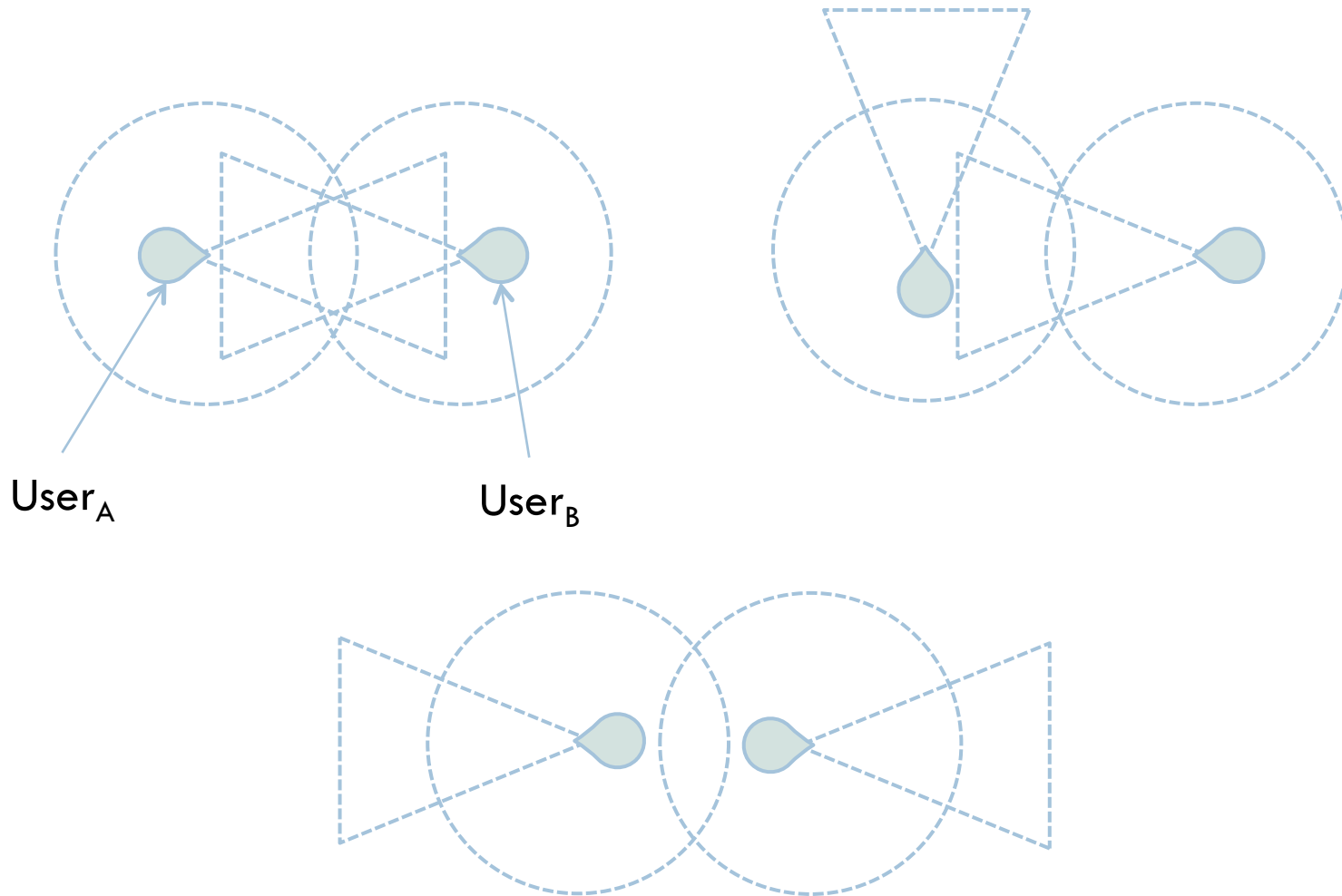
- Instead of grouping users by a global cell, group by their own interest overlap
- Aura, Focus, Nimbus (Spatial Model) pioneered in the MASSIVE and DIVE systems



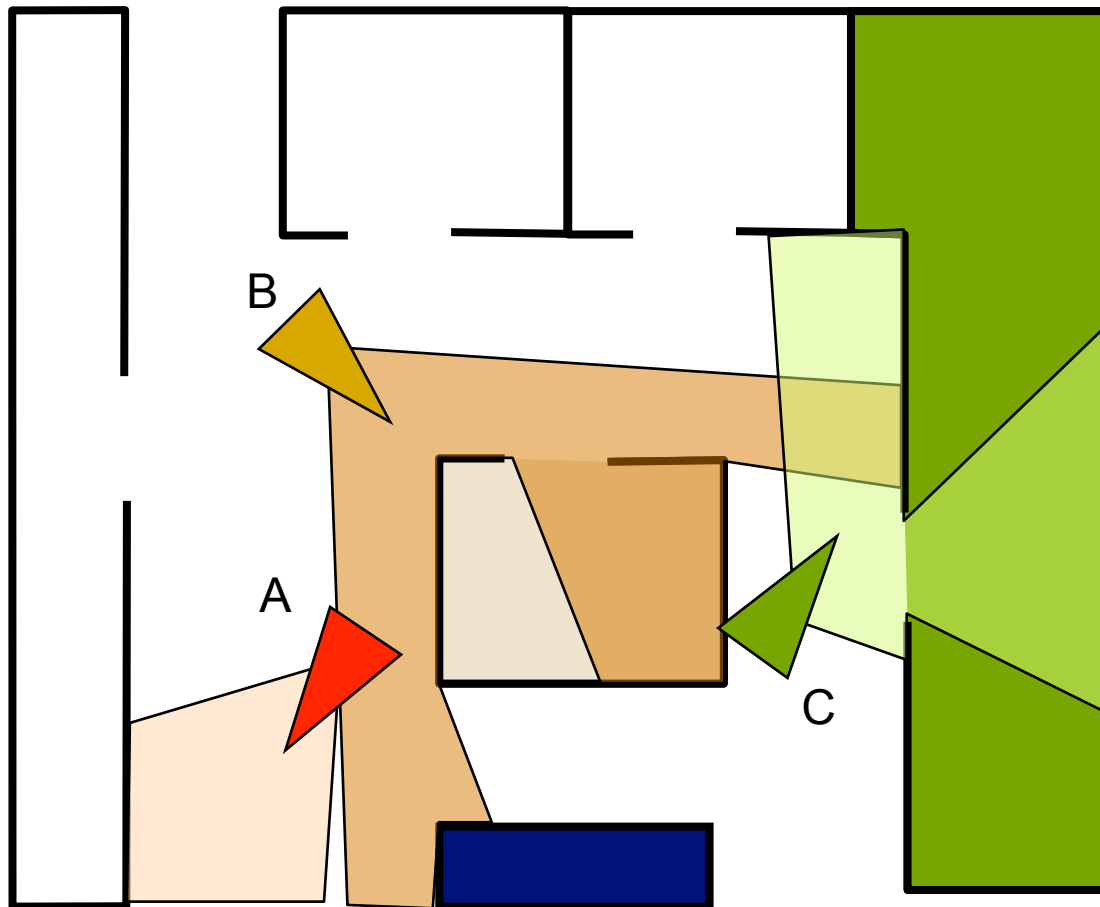
# Local Partitions: Auras



# Local Partitions: Auras



# Local Partitions: Visibility



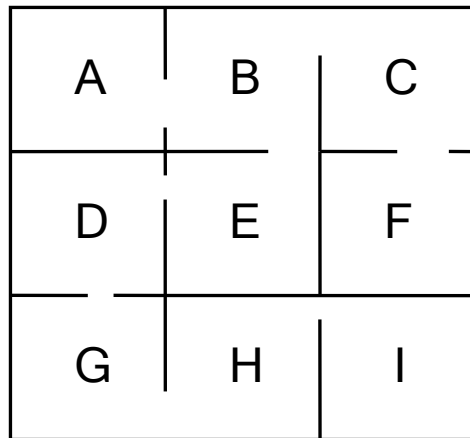
- Line of sight
- Entity visible = group
- Client/Server

# Local Partitions: Visibility

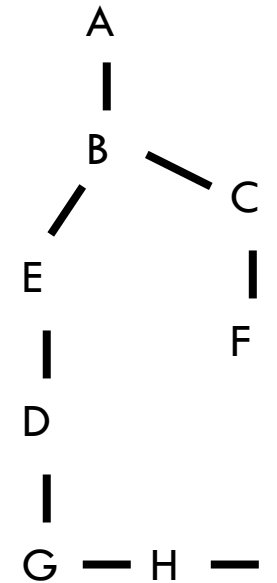


- In real environment our focus is most severely limited by the physical environment: we can't see around walls, we can't hear (or see) over long distances

# Local Partitions: Visibility

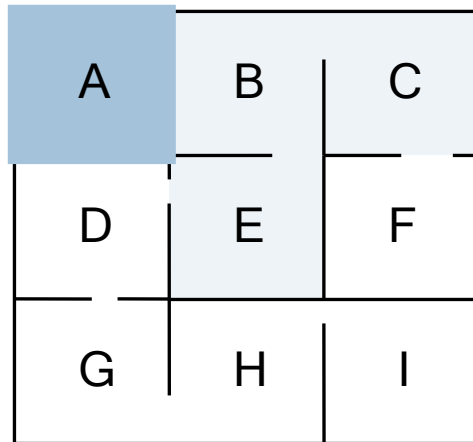


Cells



Portals

# Spatial Partitions: Visibility

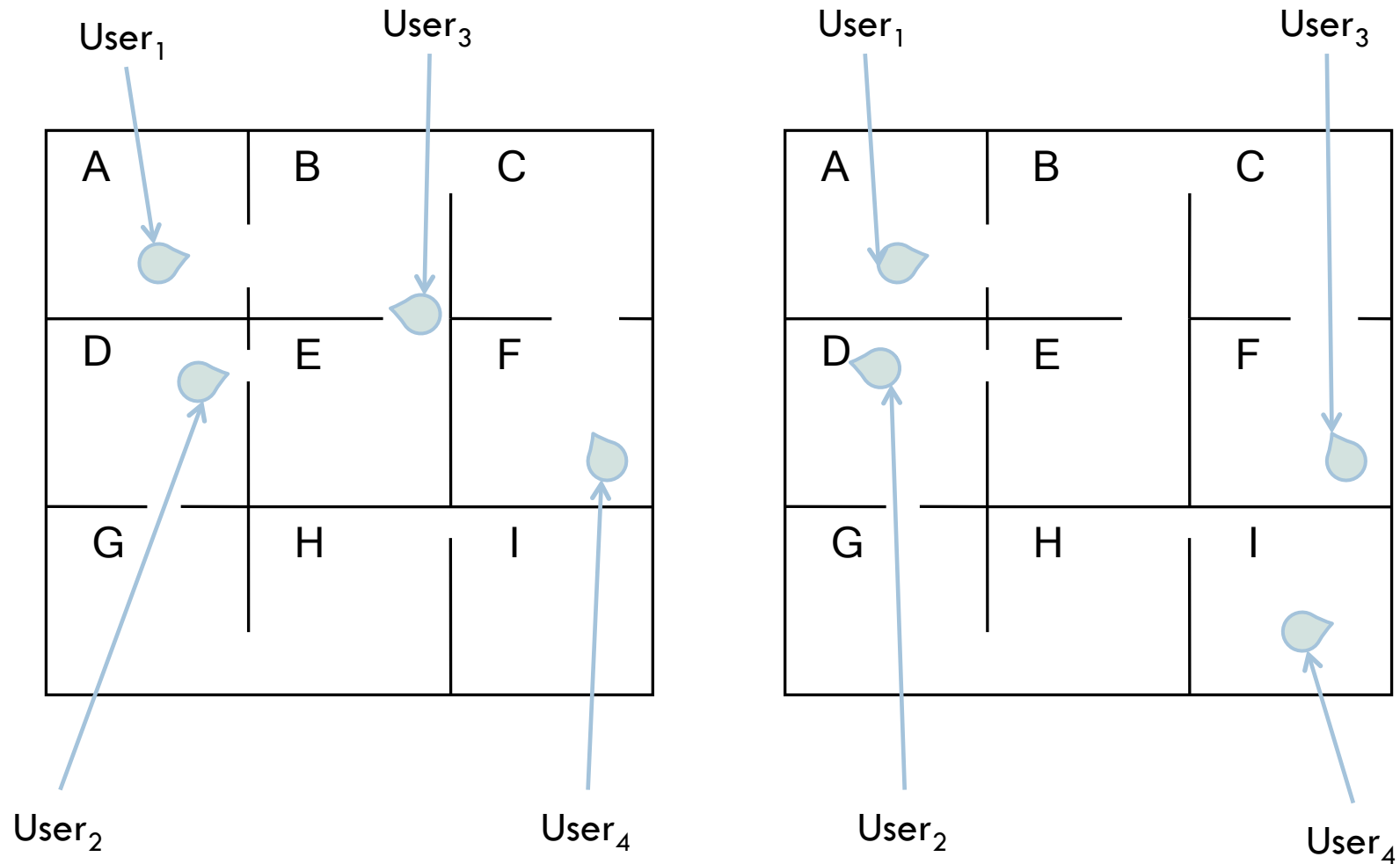


PVS<sub>A</sub>

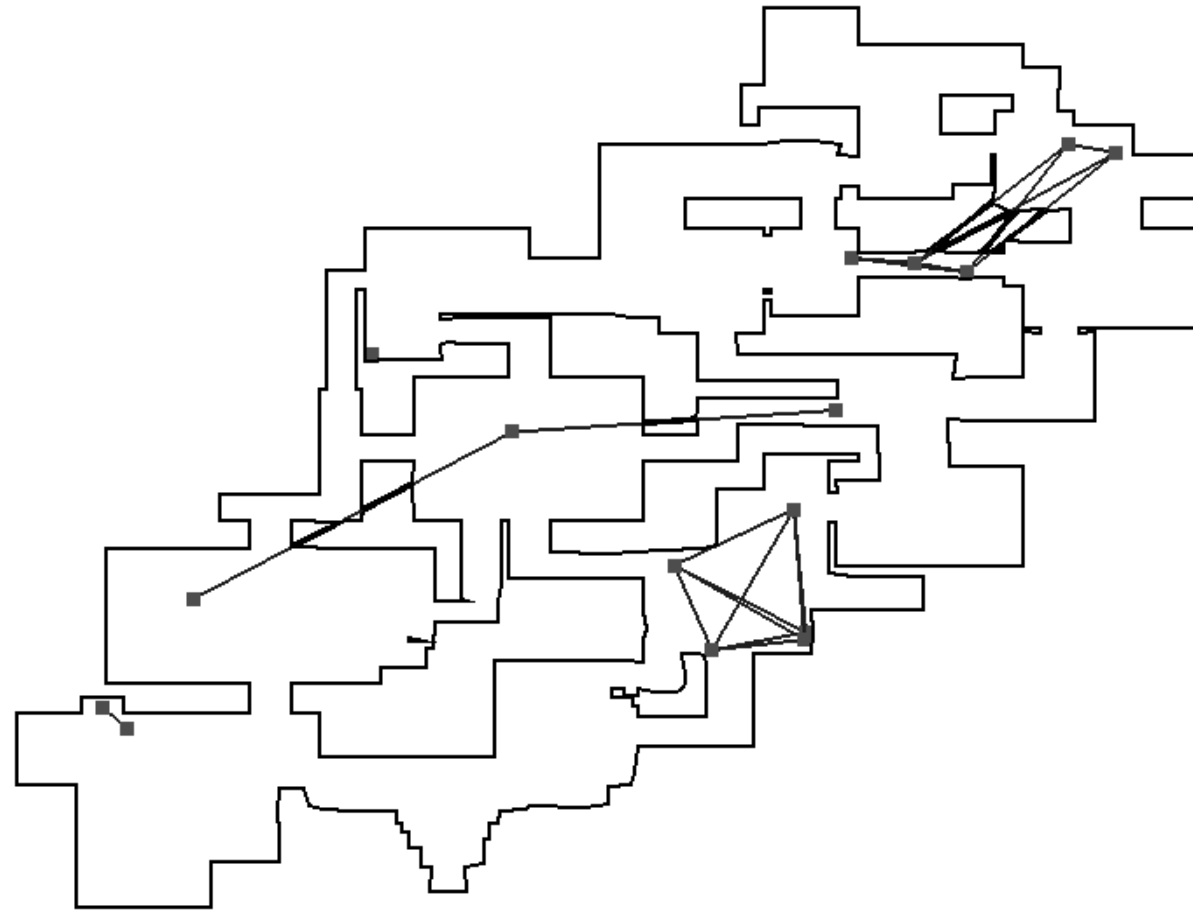
<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>	<b>F</b>	<b>G</b>	<b>H</b>	<b>I</b>	
-	<b>1</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>A</b>
	-	<b>1</b>	<b>1</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>B</b>
		-	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>C</b>
			-	<b>1</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>D</b>
				-	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>E</b>
					-	<b>0</b>	<b>0</b>	<b>0</b>	<b>F</b>
						-	<b>1</b>	<b>1</b>	<b>G</b>
							-	<b>1</b>	<b>H</b>
								-	<b>I</b>

Full PVS

# Spatial Partitions: Visibility

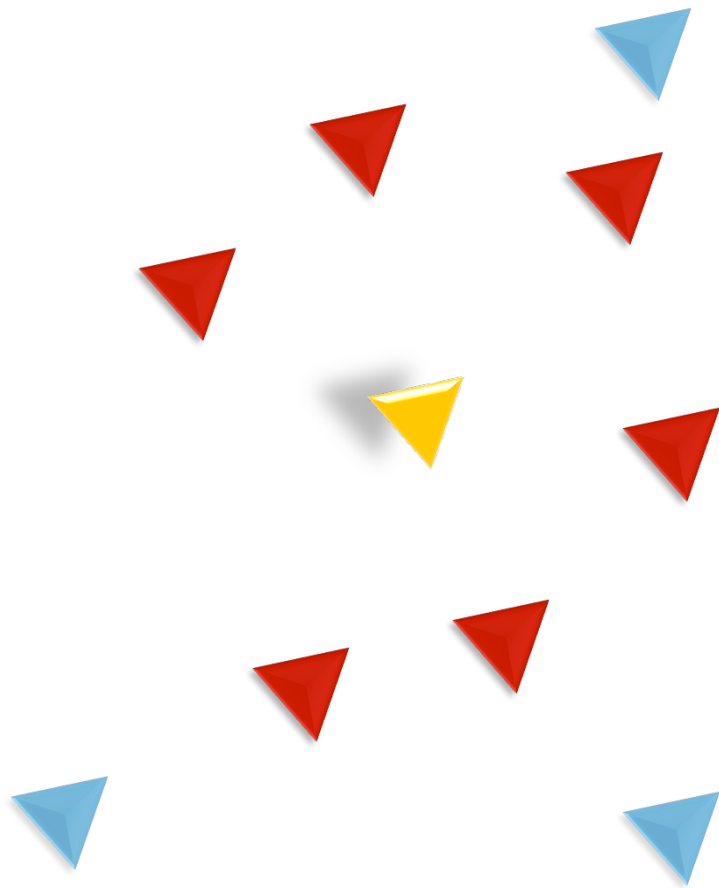


# Spatial Partitions: Visibility



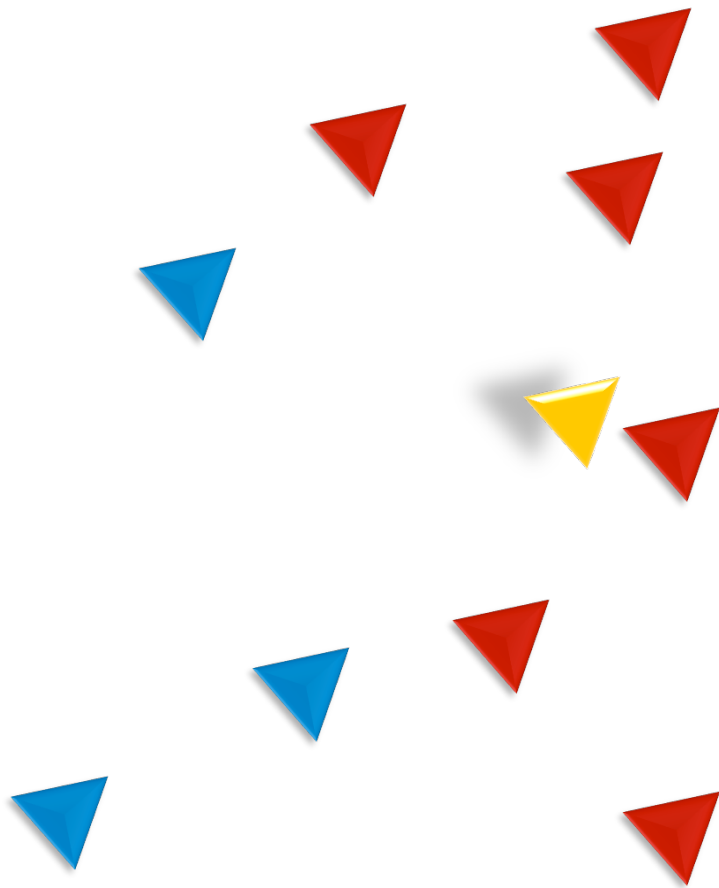


# Local Partitions: Nearest Neighbours



- 1 group = quorum
- Computational/Network constraints
- Client/Server

# Local Partitions: Nearest Neighbours



- 1 group = quorum
- Computational/Network constraints
- Client/Server



# MANAGING HANDOVER

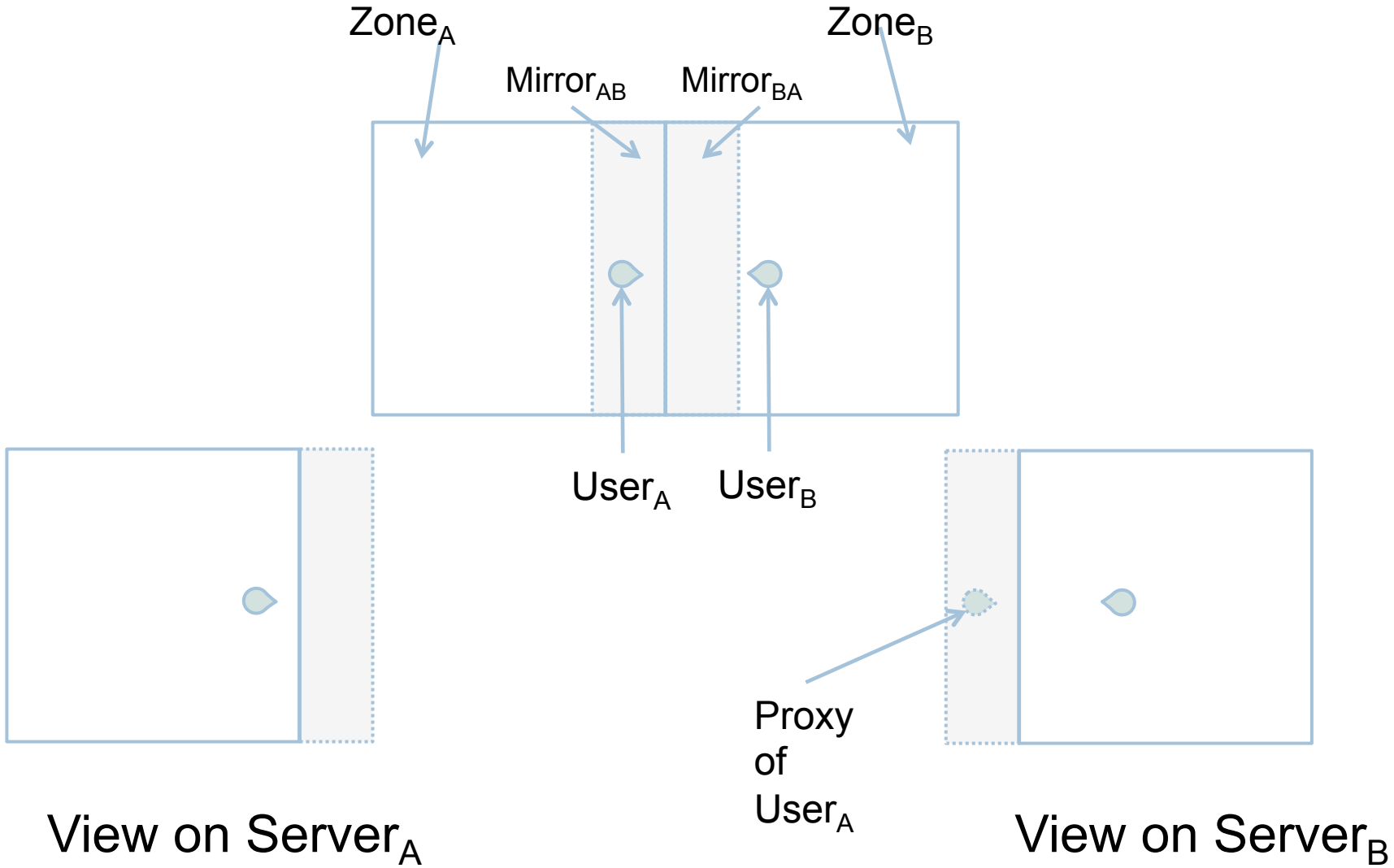


# SERVER INTERACTIONS

# Server Interactions



- Server system introduce two big problems
- How do two proximate users on adjacent servers interact?
  - ▣ Sometimes just not allowed – long twisty roads between server regions where you never meet other players
- How do you actually hand over a player from one server to another
  - ▣ Need to move responsibility for interaction
  - ▣ Possibilities needs new network connections





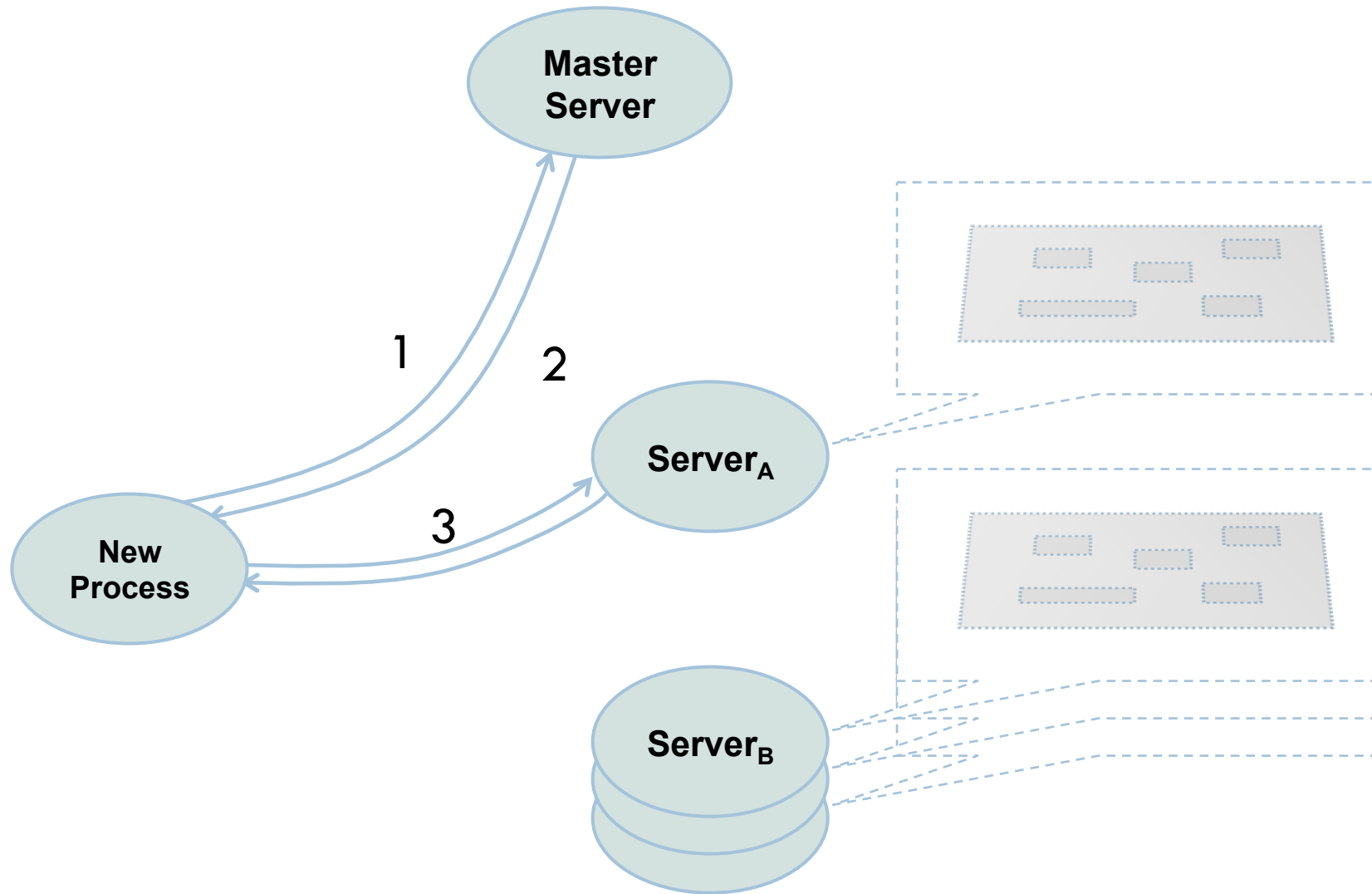
# MULTI-SERVER MANAGEMENT

# Practical Systems

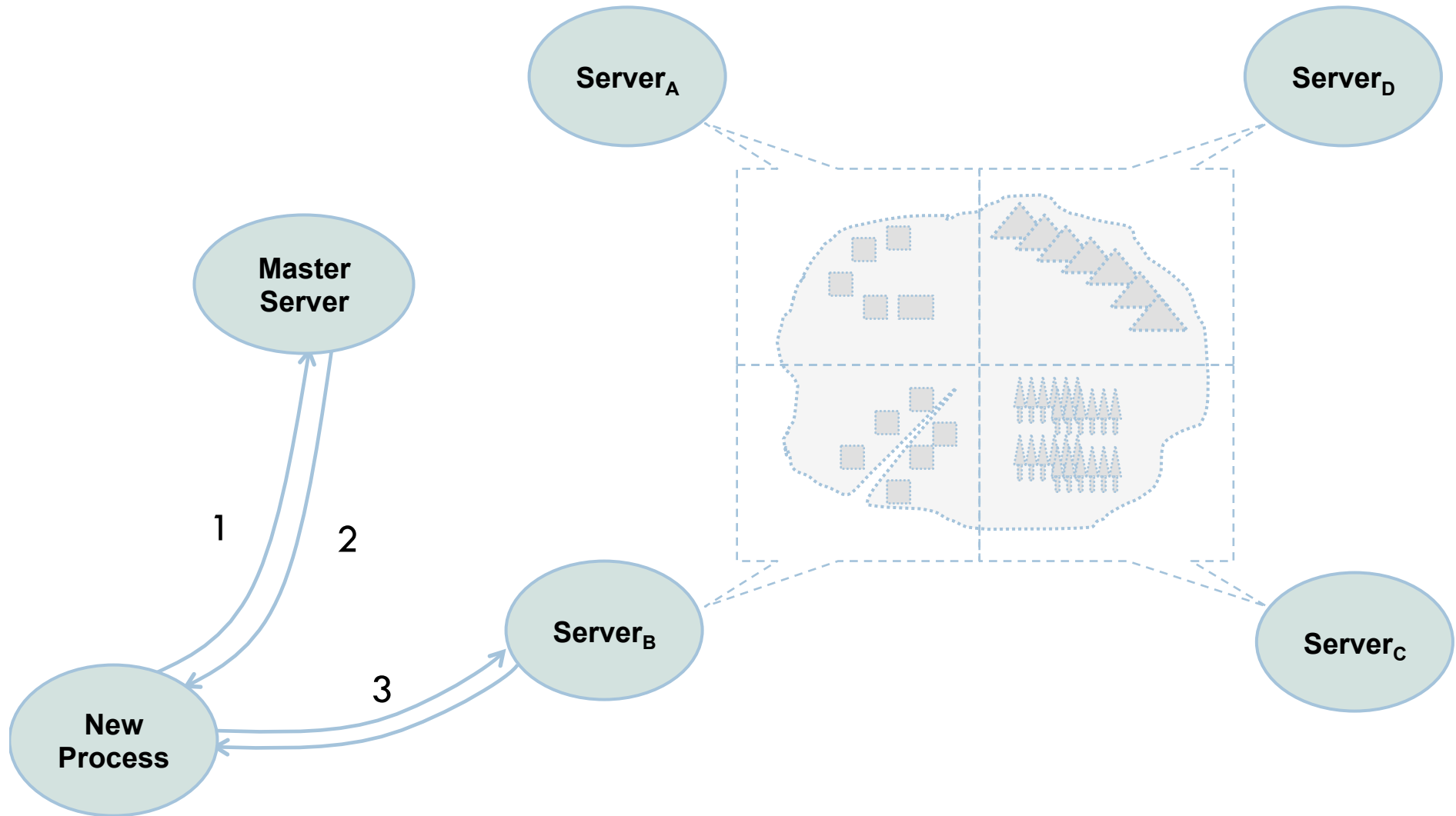
- A system such as Second Life™ utilizes a regular grid layout with one server per region
  - ▣ Regions are laid out on a mostly-contiguous map
- However in a game session, far too many players want to access a specific game content
- A game *shard* is a complete copy of a system, you connect to one 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.



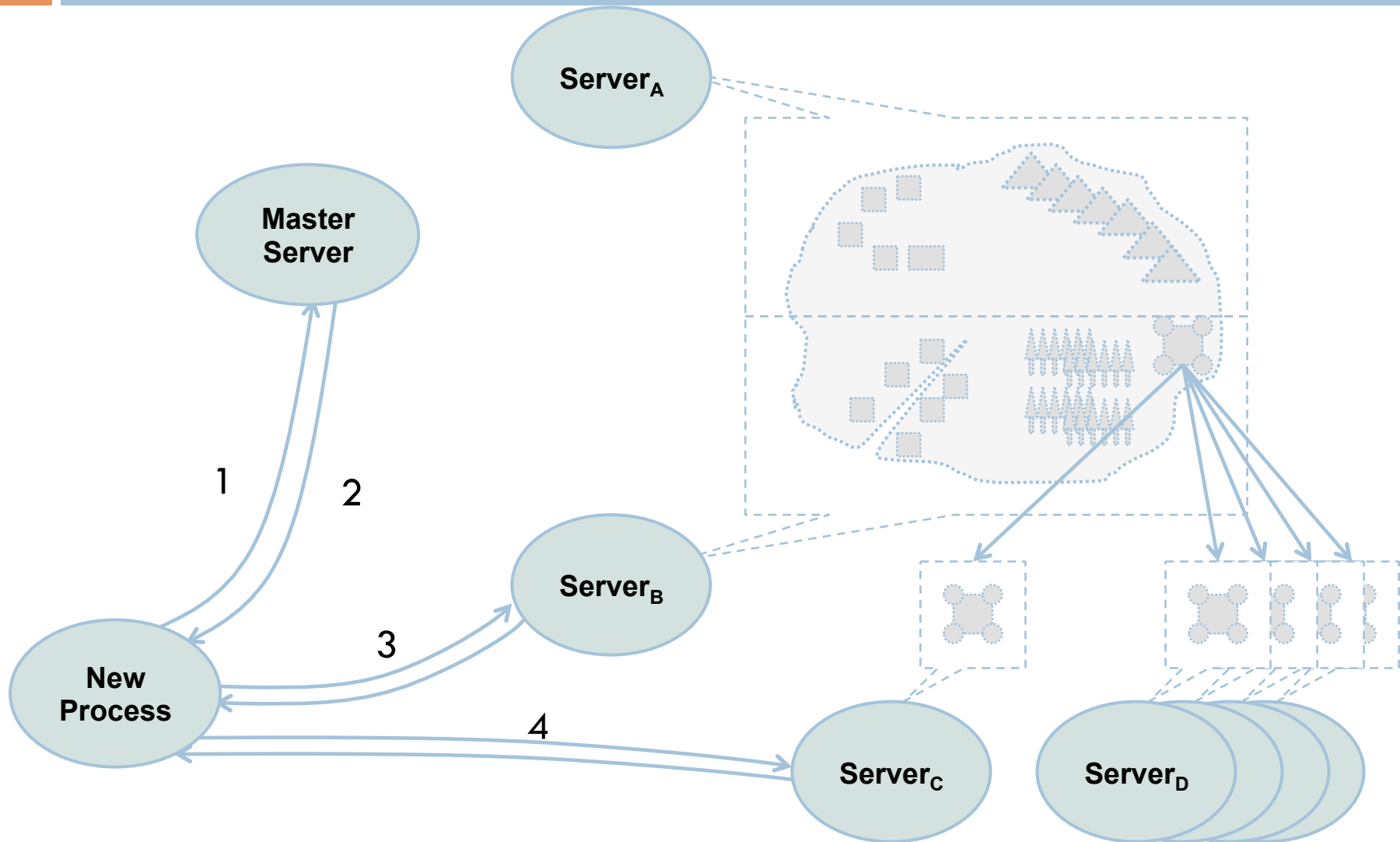
# Game Shards



# Game Regions




# Game Regions & Instances





# SUMMARY

- 
- Latency and dealing with time is a huge issue in NVEs and NGs with a variety of solutions
    - ▣ Conservative solutions v. rollback v. playout delays
    - ▣ Choice depends on game play
  - Scalability depends on a choice of awareness mechanism
    - ▣ Requires a logical scalability mechanism
    - ▣ Partitioning over users
  - Part 4 will look at application support, tools and future research issues