Algorithms and Networking for Computer Games

Chapter 9: Compensating Resource Limitations

http://www.wiley.com/go/smed

Information-centric view of resources

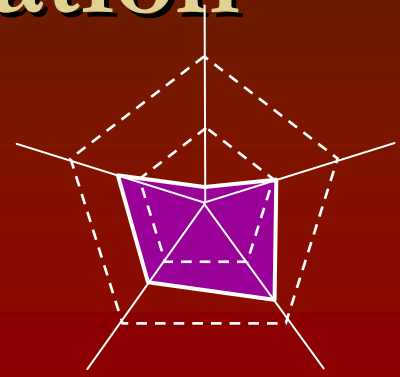
- Bandwidth requirements increase with the number of players
- Each additional player
  - must receive the initial game state and the updates that other users are already receiving
  - introduces new updates to the existing shared state and new interactions with the existing players
  - introduces new shared state
- Additional players require additional processor cycles at the existing player's host
- Each additional player
  - introduces new elements to render
  - increases the amount of caching (new shared state)
  - increases the number of updates to receive and handle

Information principle

The resource utilization is directly related to the amount of information that must be sent and received by each host and how quickly that information must be delivered by the network.

- The most scalable networked application is the one that does not require networking
- To achieve scalability and performance, the overall resource penalty incurred within a networked application must be reduced

Information principle equation



$$\text{Resources} = M \times H \times B \times T \times P$$

M = number of messages transmitted

H = average number of destination hosts for each message

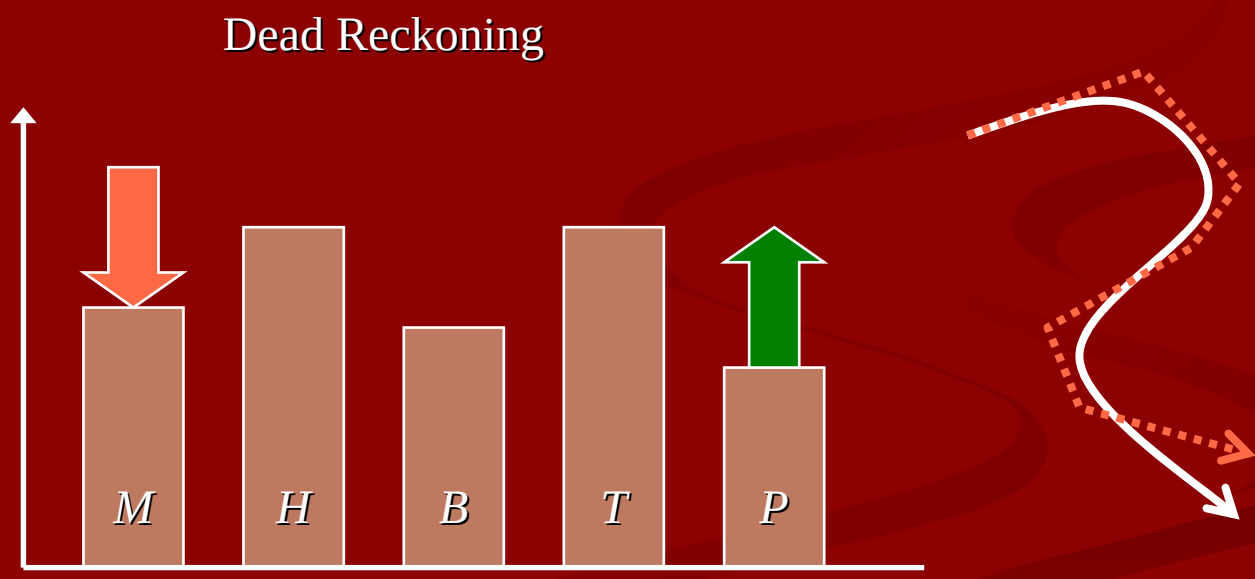
B = average amount of network bandwidth required for a message to each destination

T = timeliness in which the network must deliver packets to each destination

P = number of processor cycles required to receive and process each message

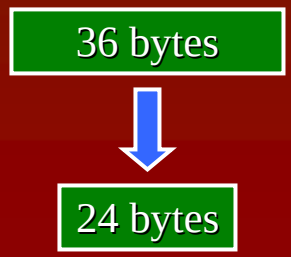
Information principle equation as a tool

- Each reduction ⇒ a compensating increase or a compensating degradation in the quality
- How to modify depends on the application

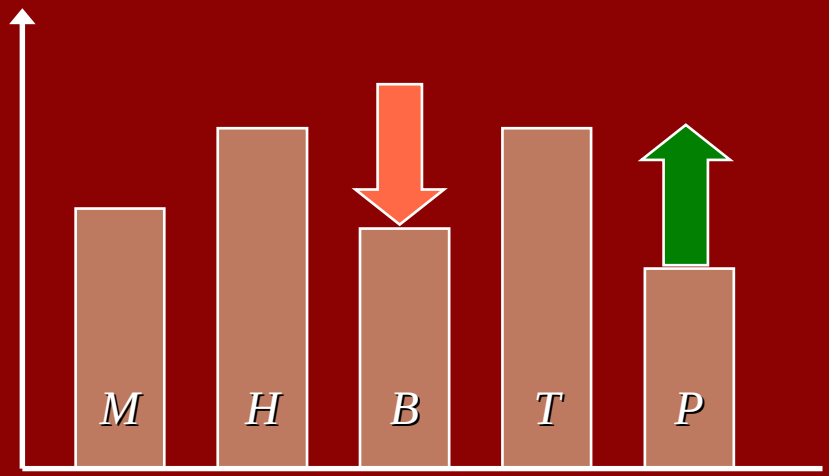


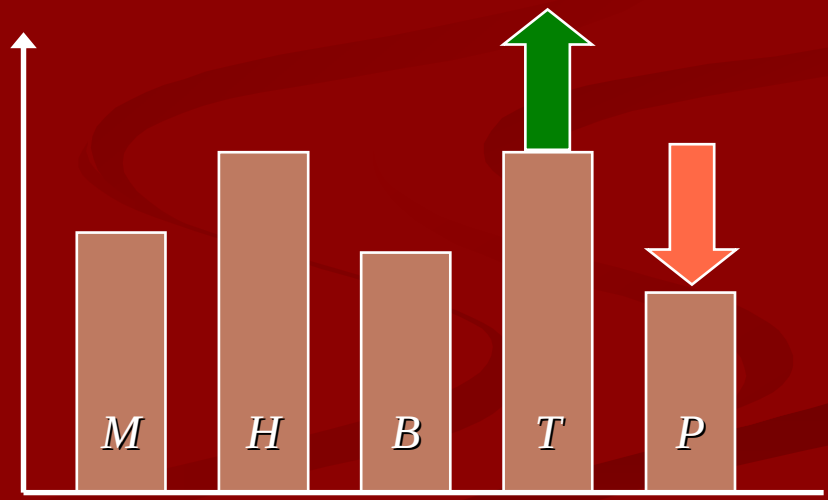
Information principle equation: examples

Message compression



Server-network





Consistency and responsiveness

- consistency
  - similarity of the view to the data in the nodes belonging to a network
- responsiveness
  - delay that it takes for an update event to be registered by the nodes
- traditionally, consistency is important
  - distributed databases
- real-time interaction ⇒ responsiveness is important and consistency can be compromised

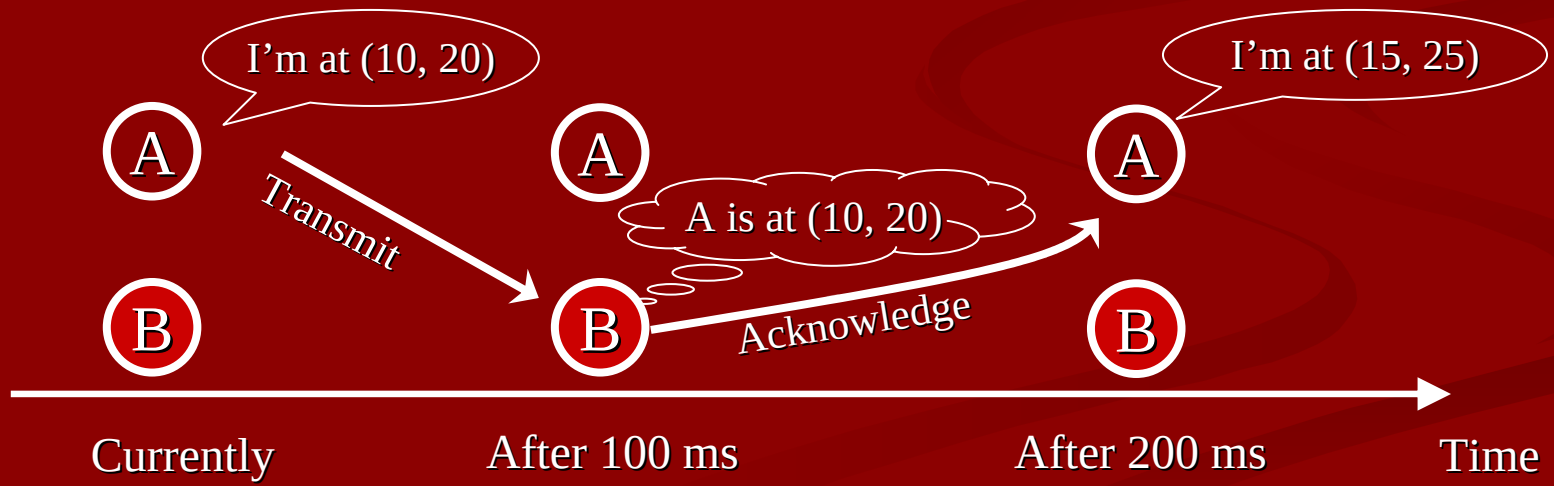
⇒ the game world can either be

- a *dynamic world* in which information changes frequently or
- a *consistent world* in which all nodes maintain identical information

but it cannot be both

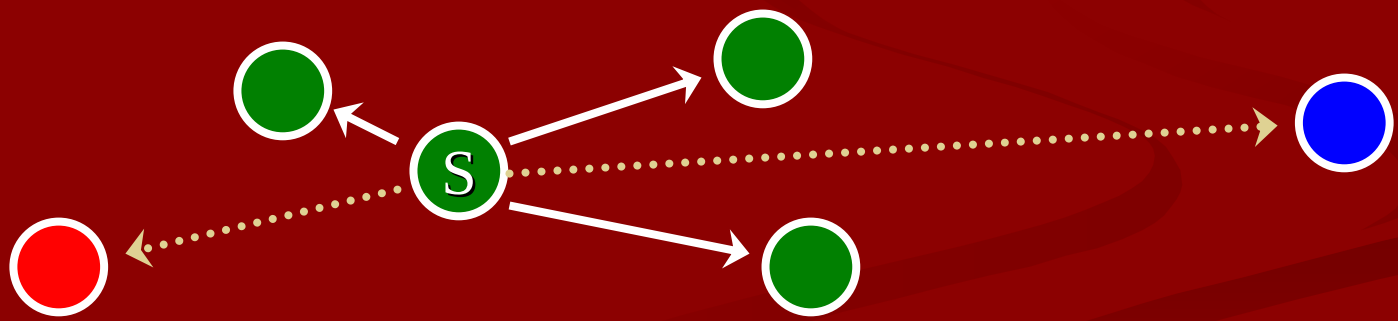
Absolute consistency

- To guarantee *absolute consistency* among the nodes, the data source must wait until everybody has received the information before it can proceed
  - delay from original message transmission, acknowledgements, possible retransmissions
- The source can generate updates only at a limited rate
- Time for the communication protocol to reliably disseminate the state updates to the remote nodes



High update rate

- There is a delay before the state change is received by other nodes
- If the state information is updated often, it might be updated while the previous update messages are still on the way
- Whilst some nodes see new values, others may still see older ones
- Because of the inherent transmission delay, one cannot update the shared state frequently and still ensure that all remote hosts have already received all previous state updates

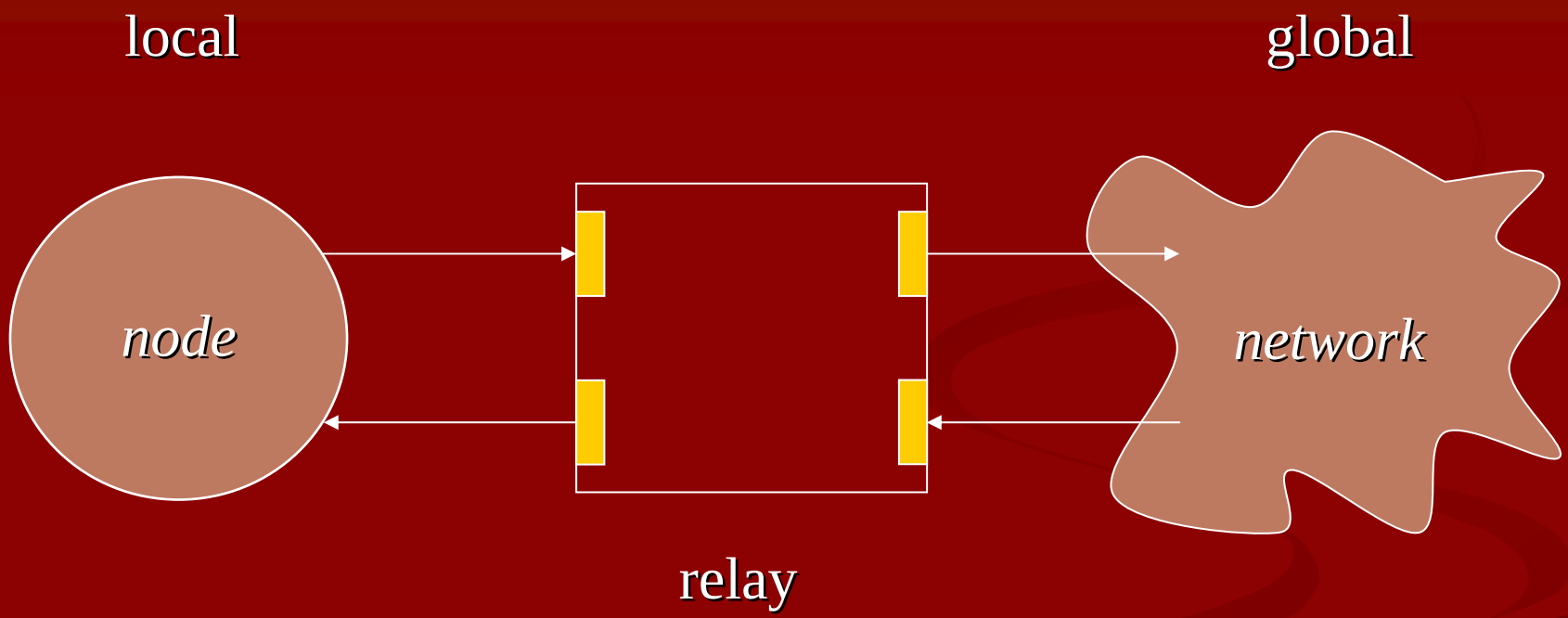


Trade-off spectrum

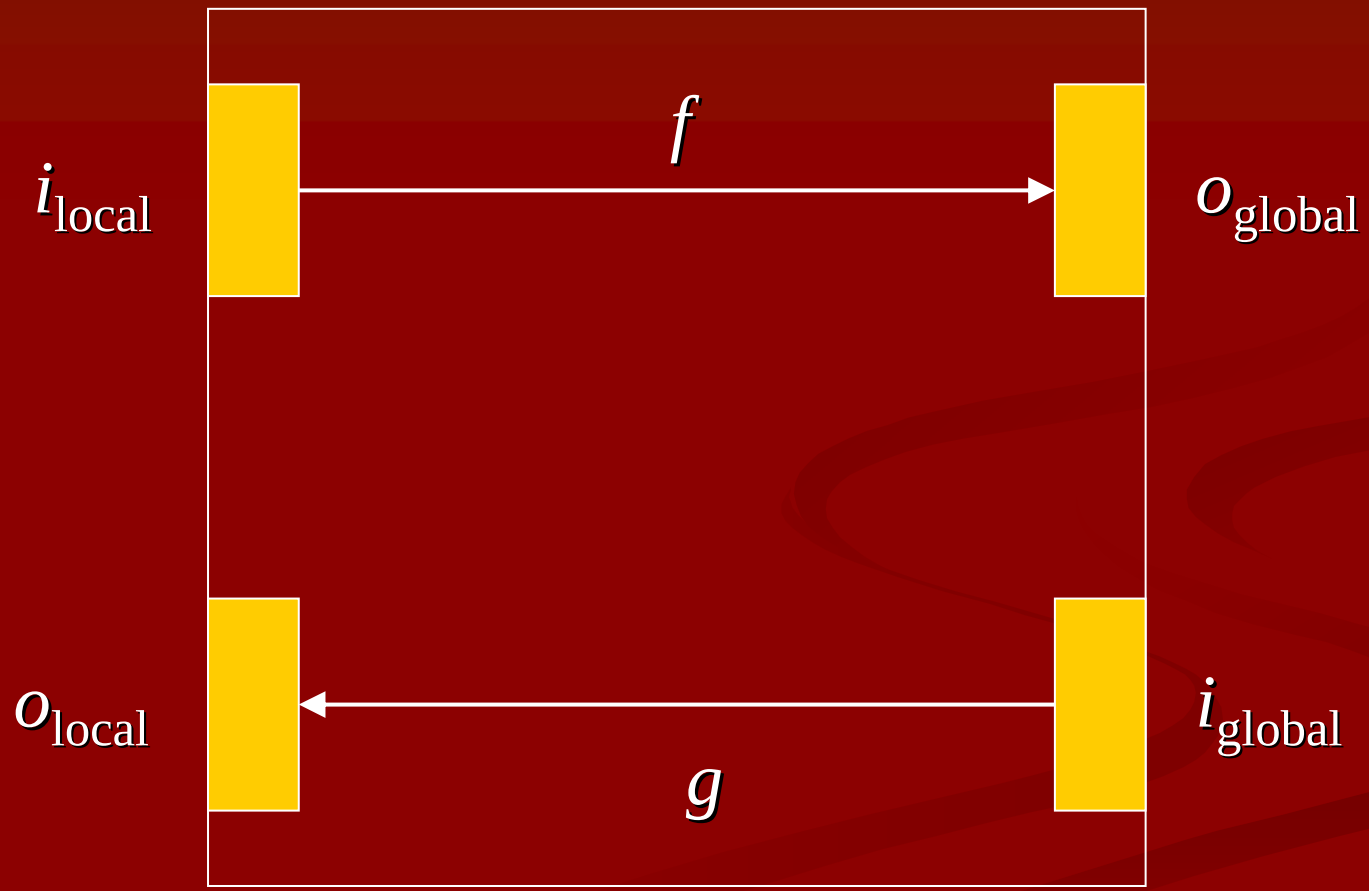
- Available network bandwidth must be allocated between
  - messages for updating the state information and
  - messages for maintaining a consistent view of the state information

  among participants.

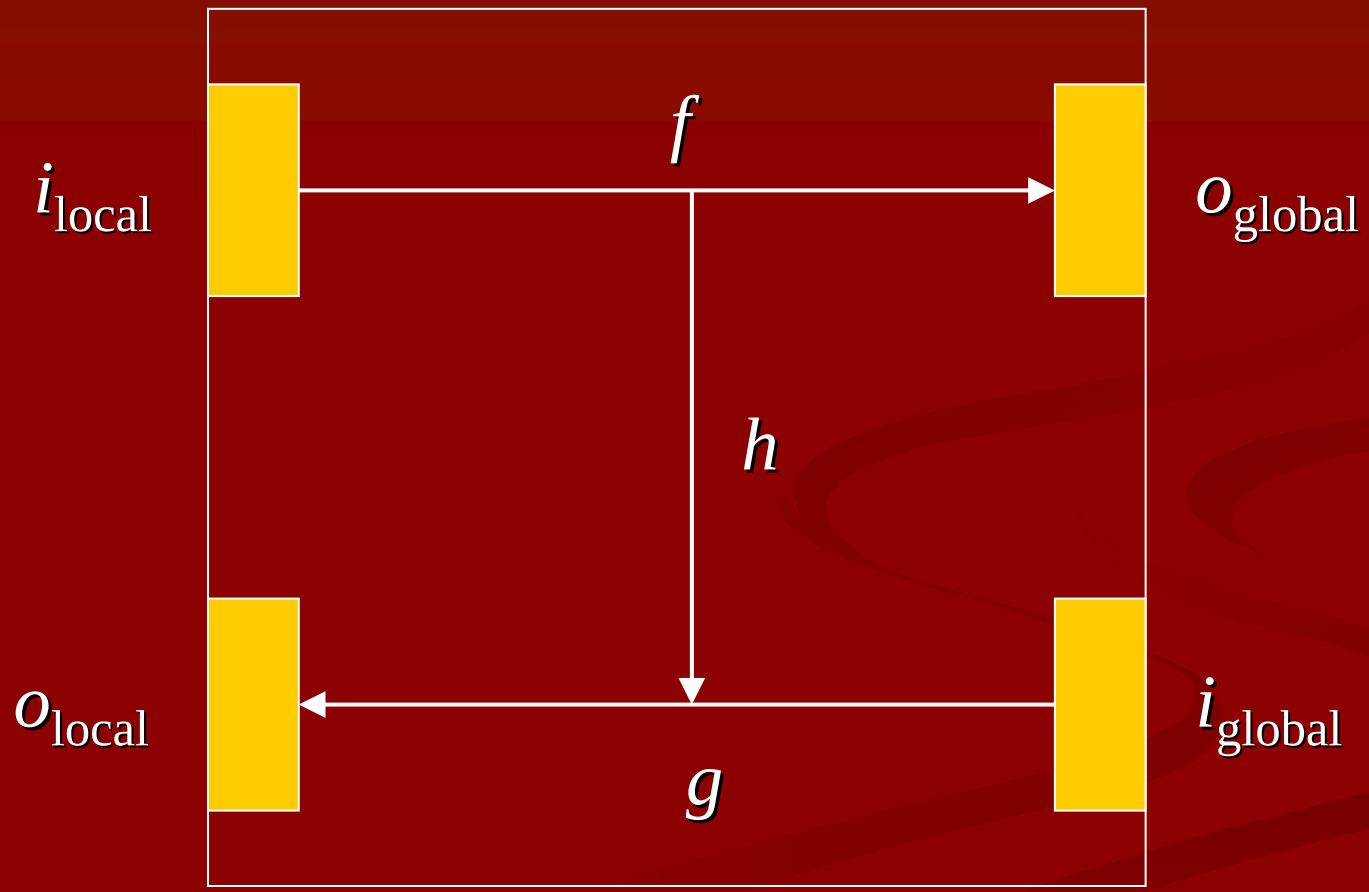
Relay model



Two-way relay



Short-circuit relay



Scalability

- Ability to adapt resource changes
  1. supporting a varying amount of human players
  2. allocating synthetic players

Amdahl's law

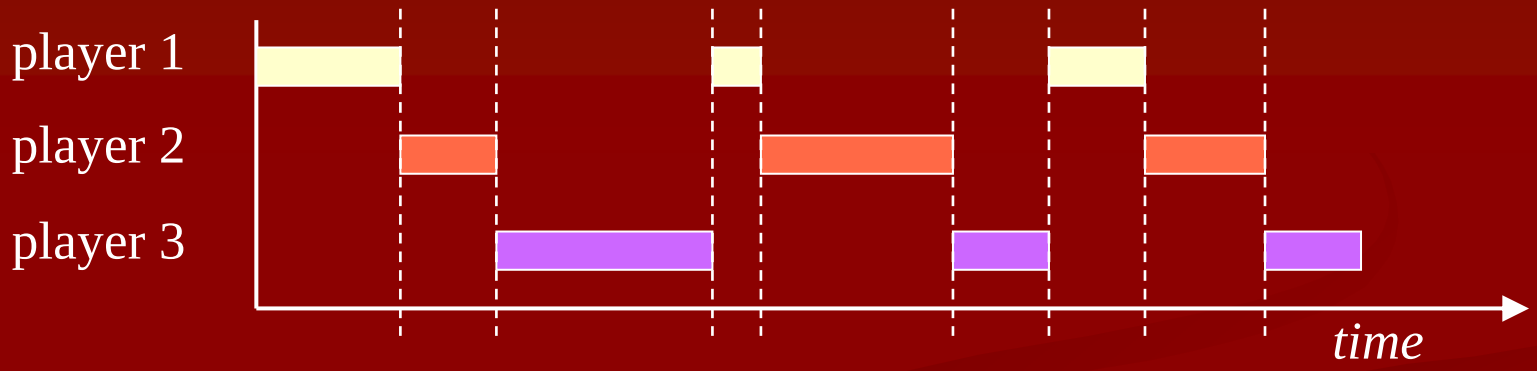
- time required by serially executed parts cannot be reduced by parallel computation
- theoretical speedup:
$$S(n) = T(1) / T(n) \leq T(1) / (T(1) / n) = n$$
- execution time has a serial part T_s and parallel part T_p
  - $T_s + T_p = 1$
  - $\alpha = T_s / (T_s + T_p)$
- speedup with optimal serialization:
$$S(n) = (T_s + T_p) / (T_s + T_p/n) \leq 1/\alpha$$
- example: $\alpha = 0.05 \Rightarrow S(n) \leq 20$

Serial and parallel execution

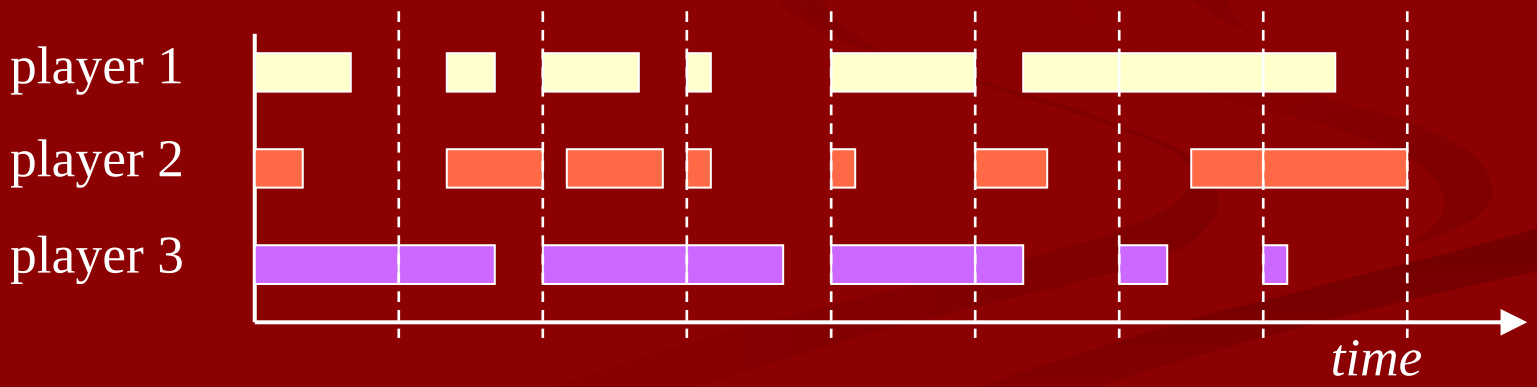
- ideally everything should be calculated in parallel
  - everybody plays their game regardless of others
- if there is communication, there are serially executed parts
  - the players must agree on the sequence of events

Interaction in a multiplayer game

Turn-based game



Real-time game



Communication capacity: example

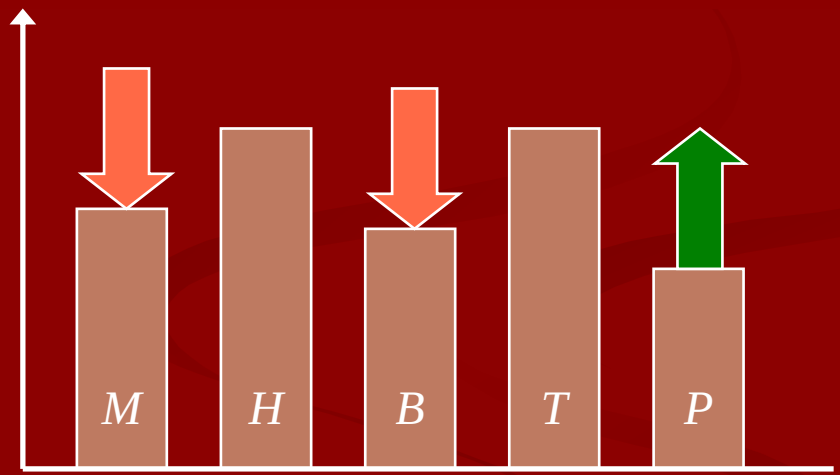
- client–server using unicasting in a 10 Mbps Ethernet using IPv6
- each client sends 5 packets/s containing a 32-bit integer value
  - bits in the message: $d = 752 + 32$
  - update frequency: $f = 5$
  - capacity of the communication channel: $C = 10^7$
  - number of unicast connections: $n = ?$
- $d \cdot f \cdot n \leq C \Rightarrow n \leq 2551$

Communication capacity

Architecture	Capacity requirement
Single node	0
Peer-to-peer	$O(n) \ldots O(n^2)$
Client–server	$O(n)$
Peer-to-peer server-network	$O(n/m + m) \ldots O(n/m + m^2)$
Hierarchical server-network	$O(n)$

Protocol optimization

- To transmit data
  - allocate a buffer
  - write data into the buffer
  - transmit a packet containing the buffer contents
- Every network packet incurs a processing penalty
- To improve resource usage, reduce
  - the size of each network packet (message compression)
  - the number of network packets (message aggregation)



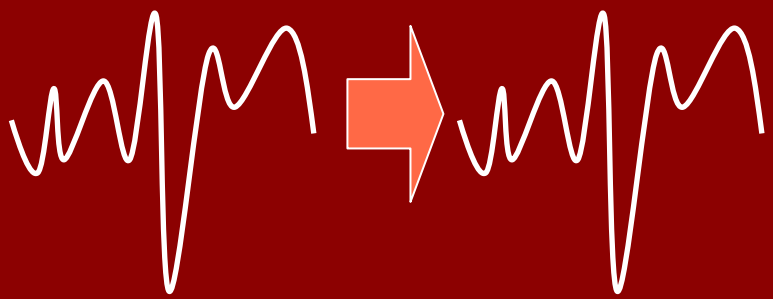
Message compression

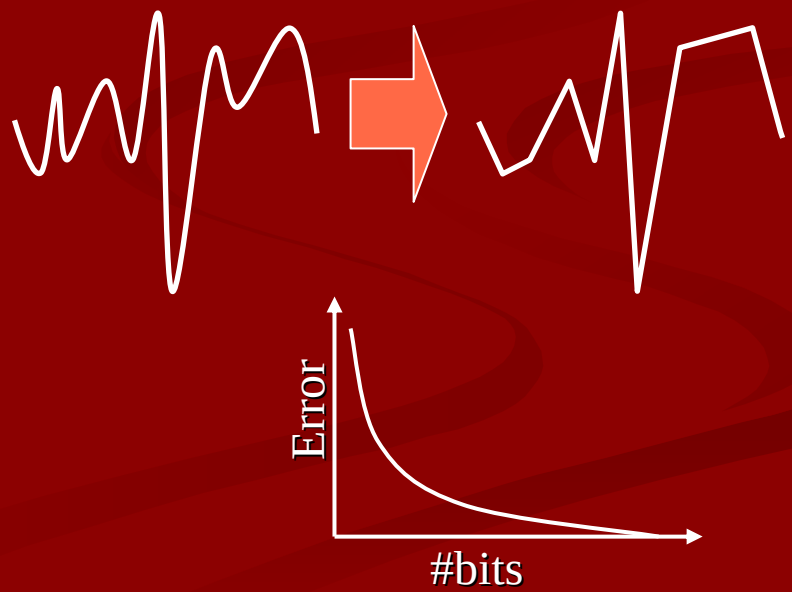
Lossless compression

- Change encoding
- No information loss
  - 10.0000001 ⇒ 10.0000001

Lossy compression

- Some information may be lost
  - 10.000000001 ⇒ 10





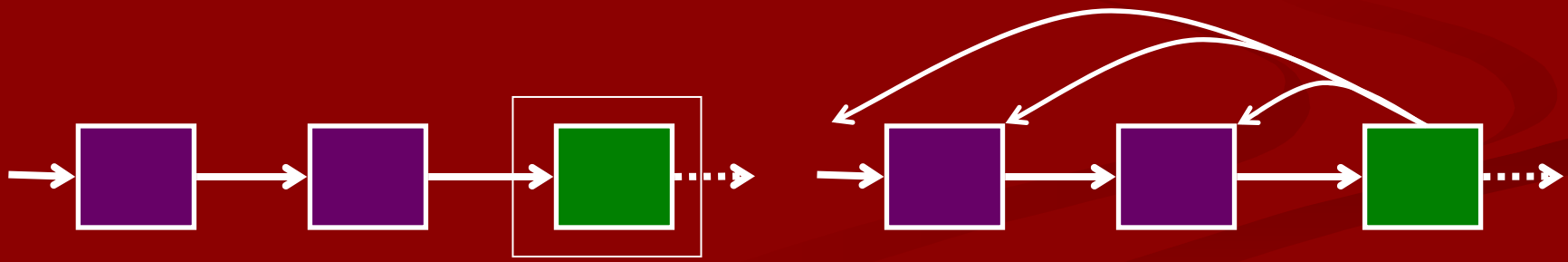
Internal and external compression

Internal compression

- Manipulates a message based solely on its own content
- No reference to the previous message

External compression

- Manipulates the message data within the context of what has already been transmitted
  - delta information
- Better compression
- Dependency between messages
- Need for reliable transmission



Compression technique categories

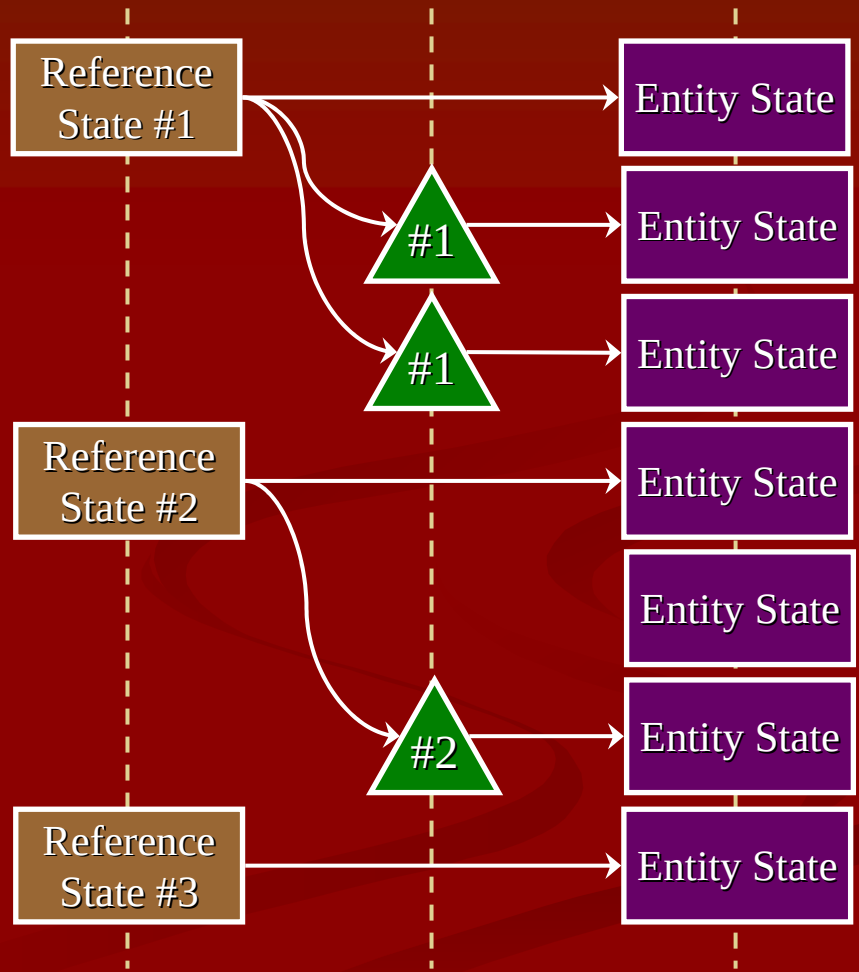
Compression technique	*Lossless compression*	*Lossy compression*
Internal compression	Encode the message in a more efficient format and eliminate redundancy within the message	Filter irrelevant information or reduce the detail of the transmitted information
External compression	Avoid retransmitting information that is identical to that sent in previous messages	Avoid retransmitting information that is similar to that sent in previous messages

Compression methods

- Huffman coding
- Arithmetic coding
- Substitutional compression
  - LZ78, LZ77
- Wavelets
- Vector quantization
- Fractal compression

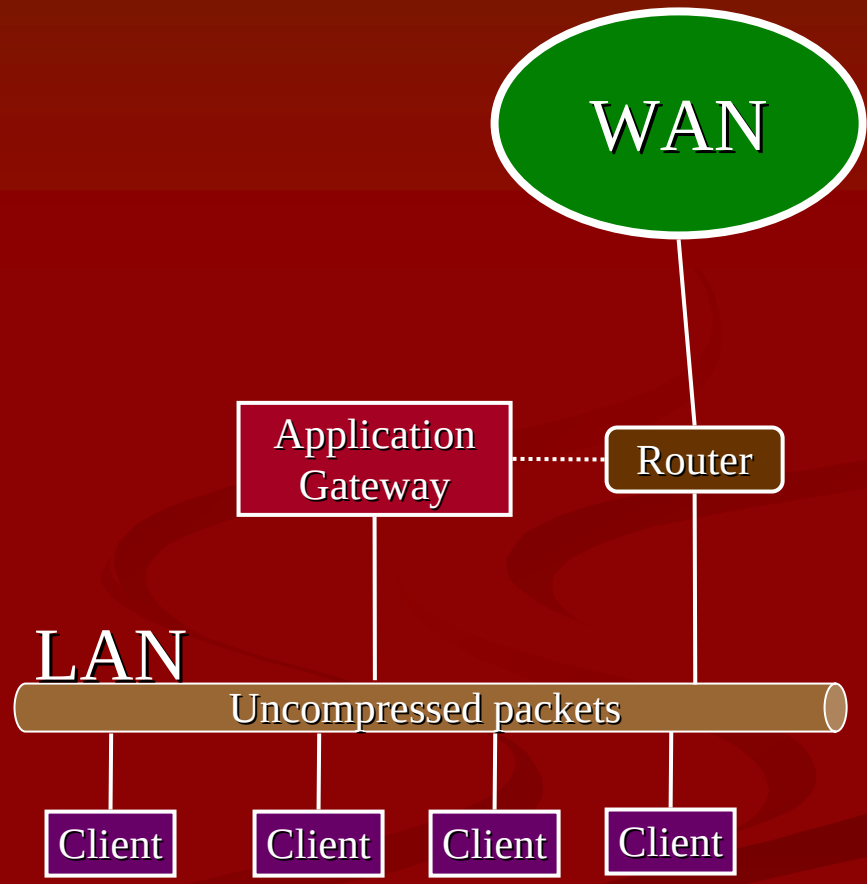
Protocol Independent Compression Algorithm (PICA)

- Lossless, external
- Transmit occasionally numbered reference state snapshots
- Subsequent update packets
  - snapshot number
  - delta information
- Snapshots reliably
  - easy retransmission



Application gateways

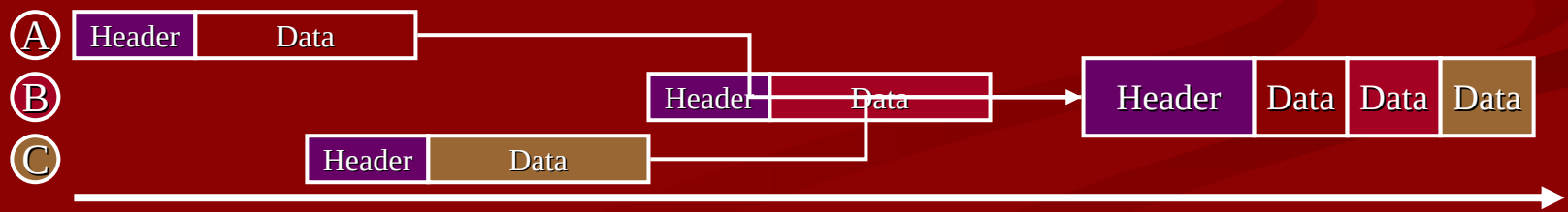
- Compression can be localized to areas of the network having limited bandwidth
- Packet in uncompressed form over the LAN
- Application Gateway (AG) compress them before they enter the WAN
- Quiescent entity service
  - handles dead or inactive entities



Message aggregation

- Reduce the number of message by merging multiple messages
- Reduces the number of headers
  - UDP/IP: 28 bytes
  - TCP/IP: 40 bytes

- Merge all messages of the local entities into a single message
  - suits when messages are transmitted at a regular frequency
  - does not decrease the quality
  - if each entity generates updates independently, the host must wait to get enough messages



Aggregation trade-offs and strategies

- Wait longer
  - better potential bandwidth savings
  - reduces the value of data
- Timeout-based transmission policy
  - collect messages for a fixed timeout period
  - guarantees an upper bound for delay
  - reduction varies depending on the entities
    - no entity updates ⇒ no aggregation but transmission delay
- Quorum-based transmission policy
  - merge messages until there is enough
  - guarantees a particular bandwidth and message rate reduction
  - no limitation on delay
- Timeliness (timeout) vs. bandwidth reduction (quorum)

Merging timeout- and quorum-based policies

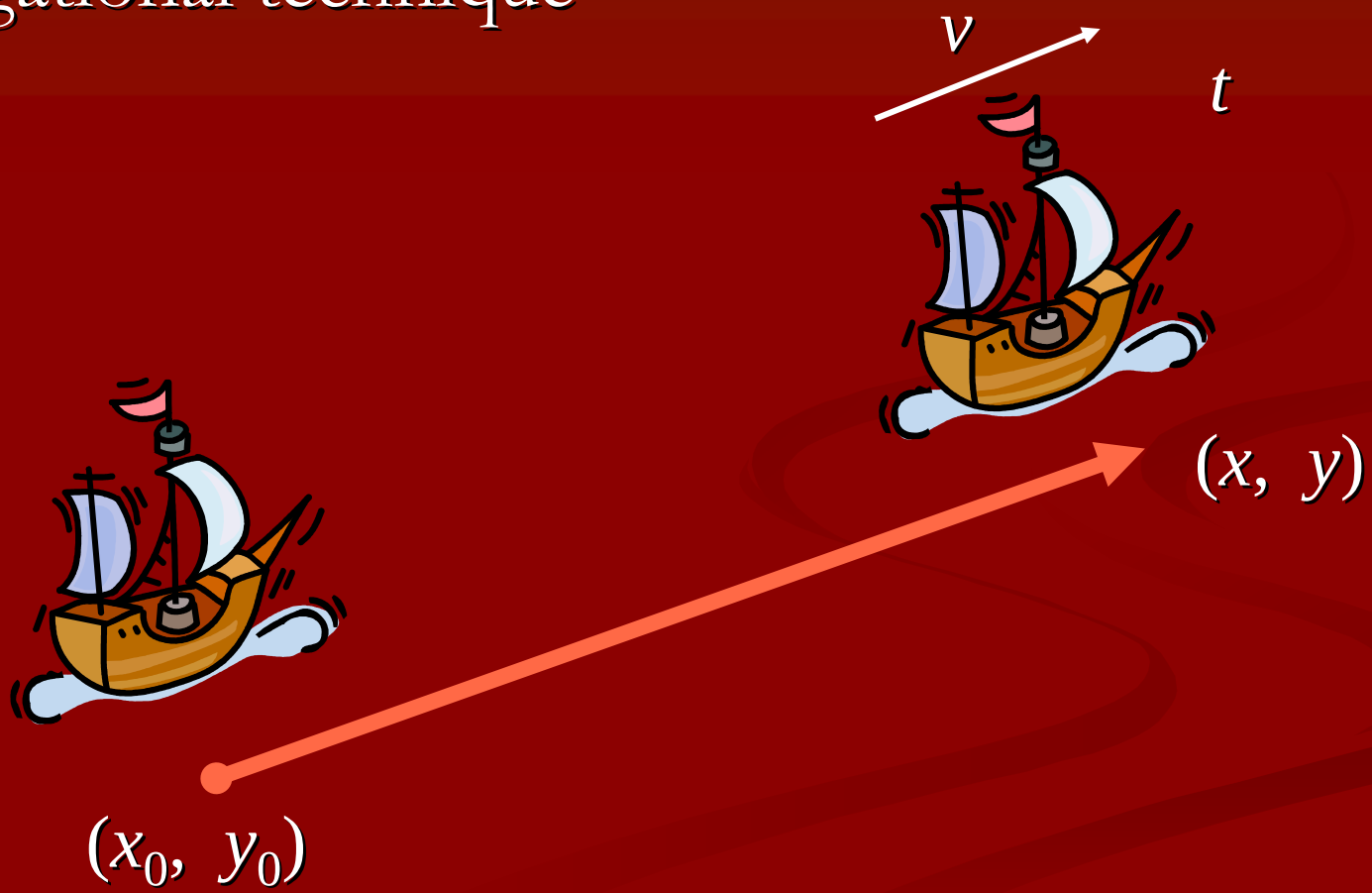
- Wait until enough messages or timeout expired
- After transmission of an aggregated message, reset timeout and message counter
- Adapts to the dynamic entity update rates
  - slow update rate ⇒ timeout bounds the delay
  - rapid update rate ⇒ better aggregation, bandwidth reduction

Aggregation servers

- In many applications, each host only manages a single entity
- More available updates, larger aggregation messages can be quickly generated
- Large update pool ⇒ projection aggregation
  - a set of entities having a common characteristic
- Aggregation server
    - location, entity type
  - hosts transmit updates to aggregation server(s)
  - server collects updates from multiple hosts
  - server disseminates aggregated update messages
- Distributes the workload across several processors
- Improves fault tolerance and overall performance

Dead reckoning

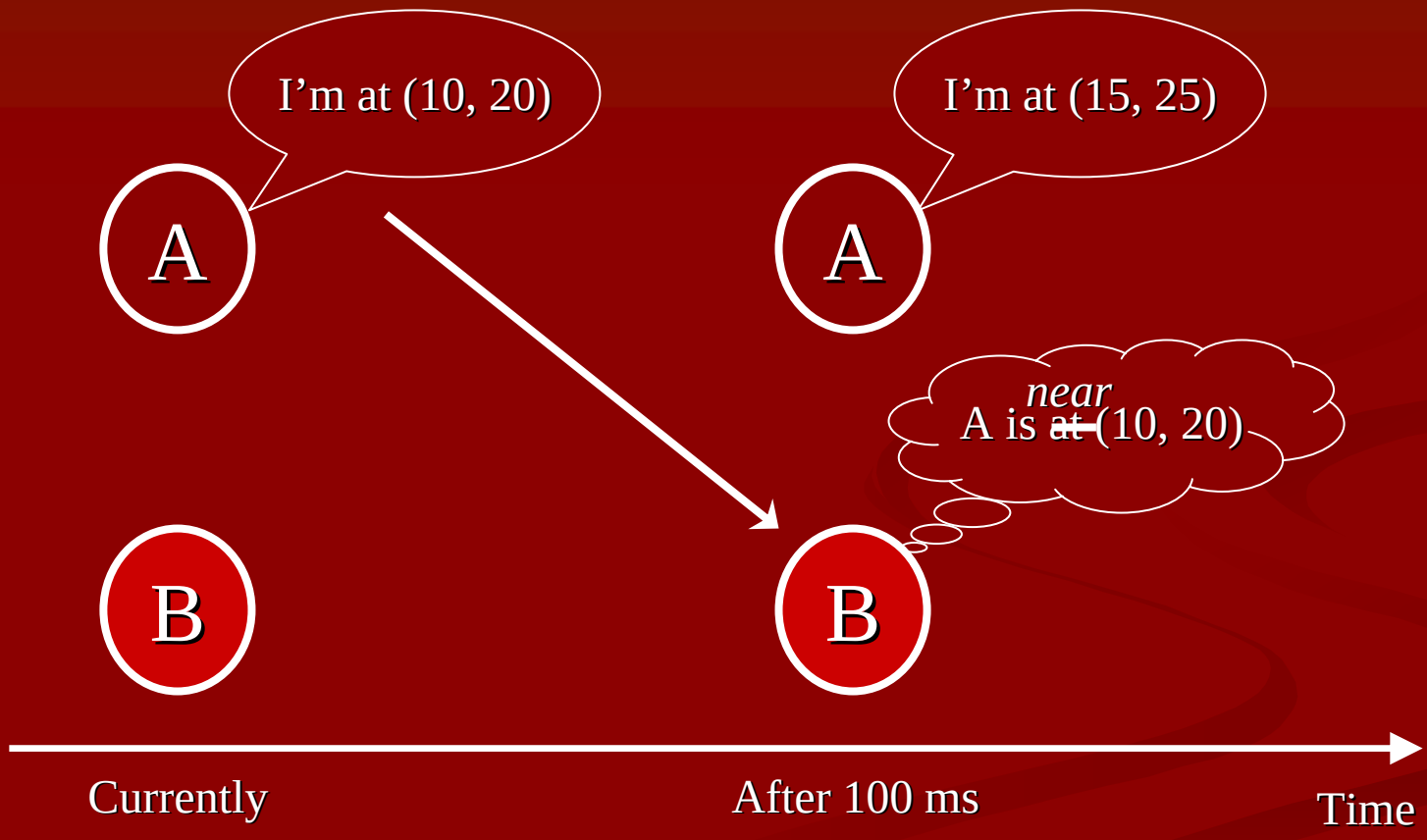
- navigational technique



Dynamic shared state

- Dynamic shared state constitutes the changing information that multiple nodes must maintain
  - participants, their locations and behaviours
  - environment itself, all objects, weather, natural laws,...
- In a highly dynamic environment, almost all information about the game world may change ⇒ needs to be shared
- Accuracy is fundamental to creating realistic environments
- Makes an environment available to multiple users
  - without dynamic shared state, each user works independently (and alone)

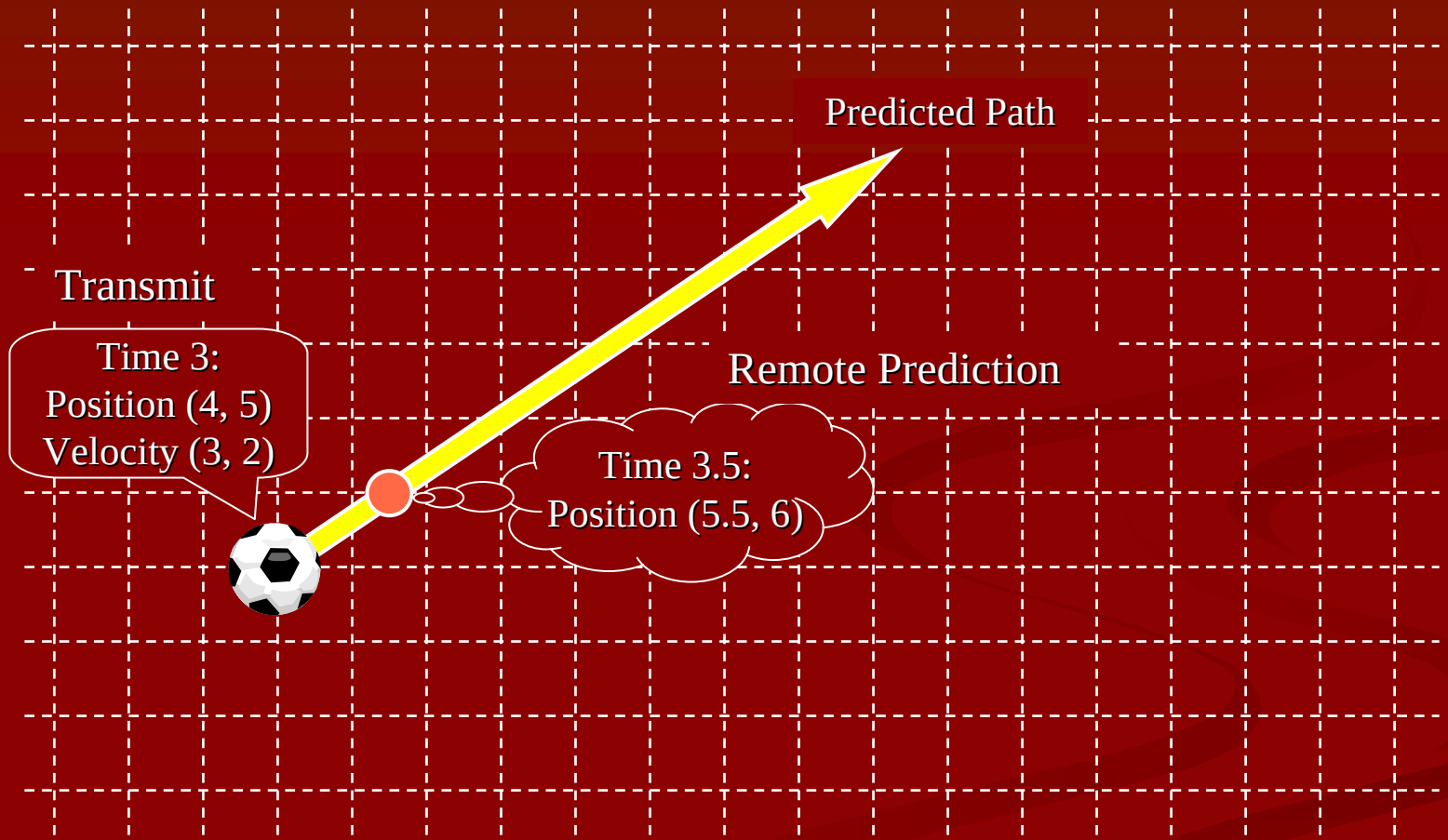
Example of dynamic shared state



Dead reckoning of shared state

- Transmit state update packets less frequently
- Use received information to *approximate* the true shared state
- In between updates, each node predicts the state of the entities

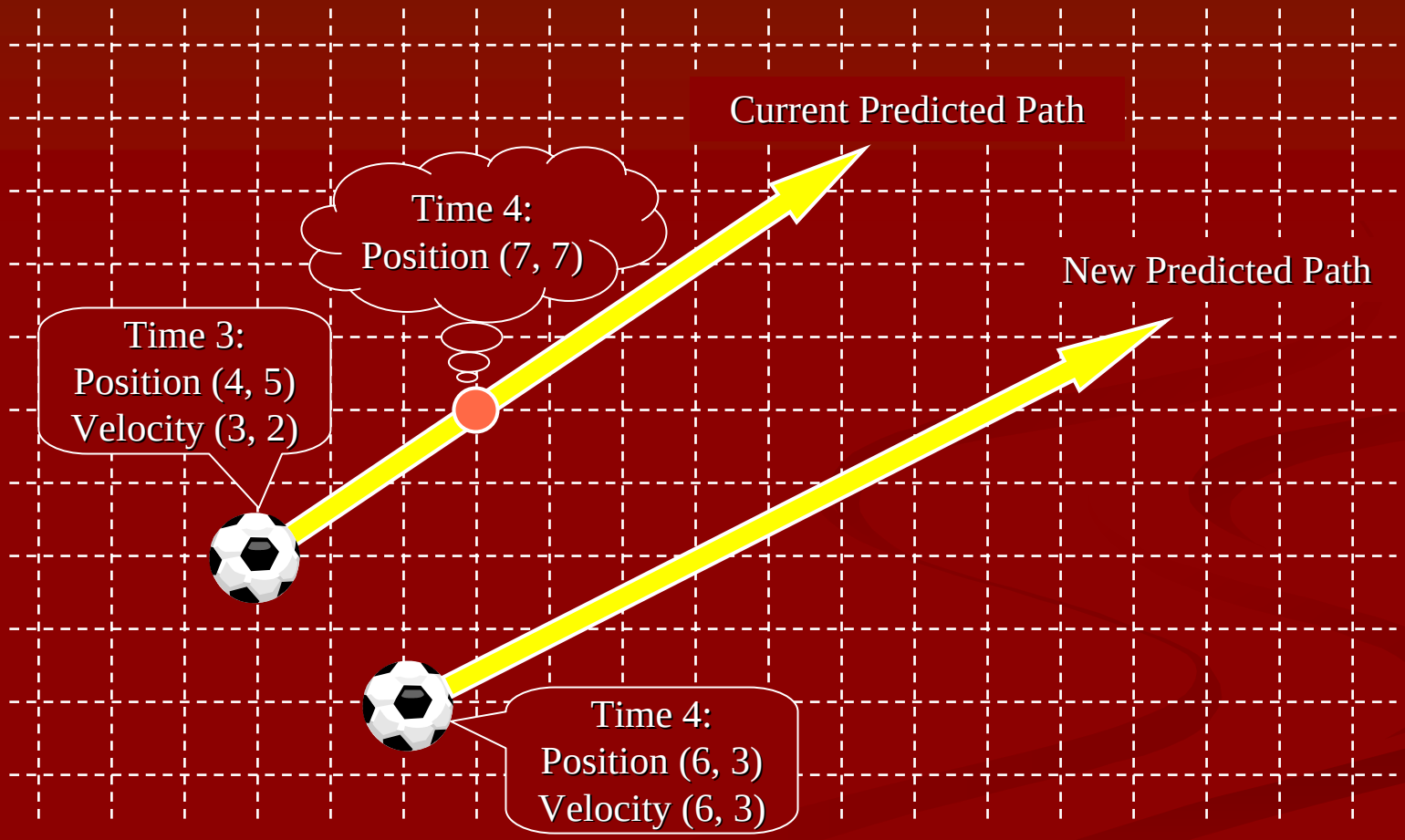
Dead reckoning: example



Dead reckoning protocol

- prediction technique
  - how the entity's current state is computed based on previously received update packets
- convergence technique
  - how to correct the state information when an update is received

Prediction and convergence



Prediction using derivative polynomials

- The most common DR protocols use derivative polynomials
- Involves various derivatives of the entity's current position
- Derivatives of position
  1. velocity
  2. acceleration
  3. jerk

Zero-order and first-order polynomials

- Zero-order polynomial
  - position p
  - the object's instantaneous position, no derivative information
  - predicted position after t seconds = p
- First-order polynomial
  - velocity v
  - predicted position after t seconds = $vt + p$
  - update packet provides current position and velocity

Second-order polynomials

- We can usually obtain better prediction by incorporating more derivatives
- Second-order polynomial
  - acceleration a
  - predicted position after t seconds
    $= \frac{1}{2}at^2 + vt + p$
  - update packet: current position, velocity, and acceleration
  - popular and widely used
  - easy to understand and implement
  - fast to compute
  - relatively good predictions of position

Hybrid polynomial prediction

- The remote host can dynamically choose the order of prediction polynomial
  - first-order or second-order?
- First-order
  - fewer computational operations
  - good when acceleration changes frequently or when acceleration is minimal
  - prediction can be more accurate without acceleration information

Position History-Based Dead Reckoning

- Chooses dynamically between first-order and second-order
- Evaluates the object's motion over the three most recent position updates
- If acceleration is minimal or substantial, use first-order
  - threshold cut-off values for each entity
- The acceleration behaviour affects to the convergence algorithm selection
- Ignores instantaneous derivative information
  - update packets only contain the most recent position
  - estimate velocity and acceleration
- Reduces bandwidth requirement
- Improves prediction accuracy in many cases

Limitations of derivative polynomials

- Add more terms to the derivative polynomial—why not?
- With higher-order polynomials, more information have to be transmitted
- The computational complexity increases
  - each additional term requires few extra operations
- Sensitivity to errors
  - derivative information must be accurate
  - inaccurate values for the higher derivatives might actually make the prediction worse

$$p(t) = \tfrac{1}{2}at^2 + vt + p$$

Limitations of derivative polynomials (cont'd)

- Hard to get accurate instantaneous information
  - entity models typically contain velocity and acceleration
  - higher-order derivatives must be estimated or tracked
  - defining jerk (change in acceleration):
    - predict human behaviour
    - air resistance, muscle tension, collisions,…
  - values of higher-order derivatives tend to change more rapidly than lower-order derivatives

⇒High-order derivatives should generally be avoided

- The Law of Diminishing Returns
  - more effort typically provides progressively less impact on the overall effectiveness of a particular technique

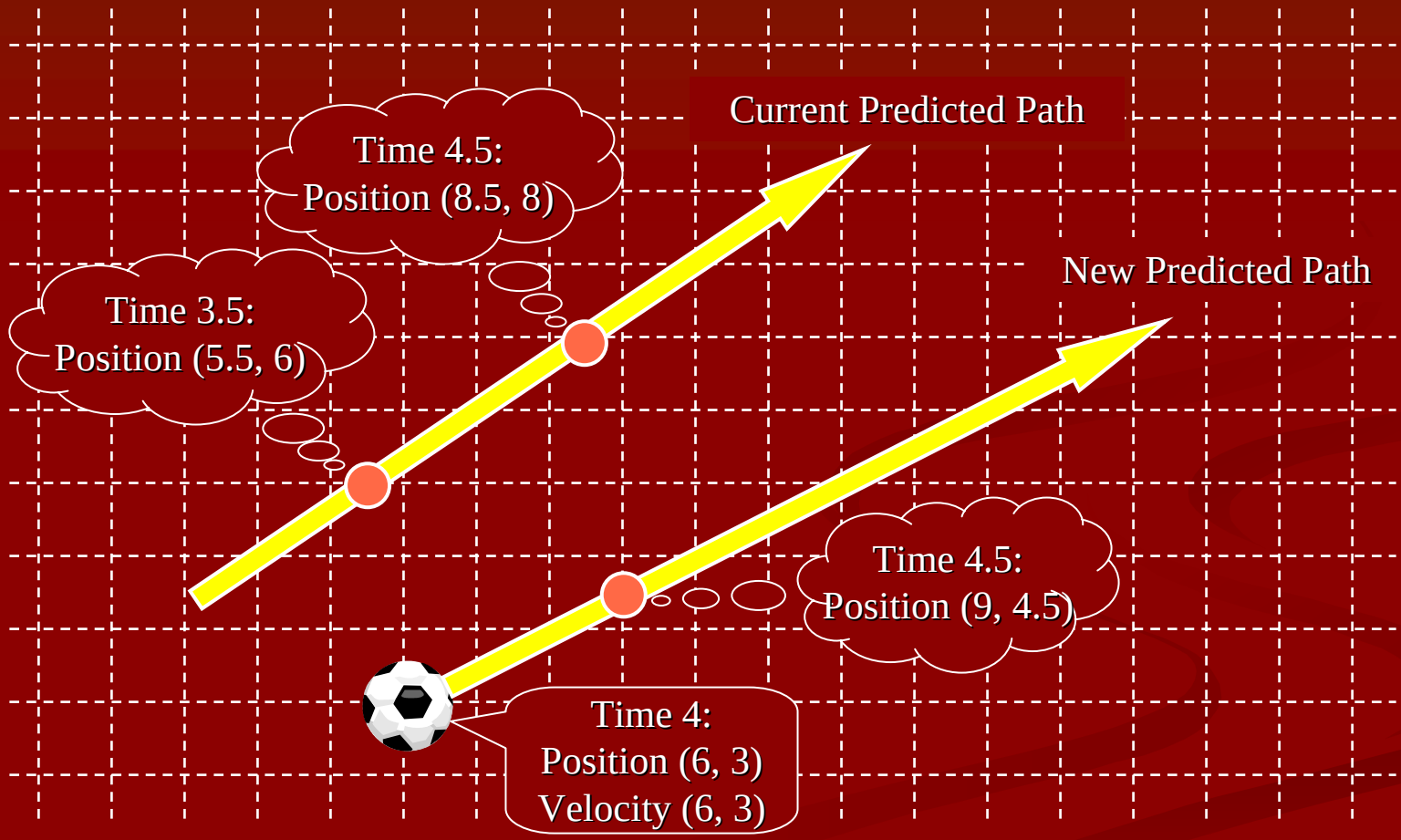
Object-specialized prediction

- Derivative polynomials do not take into account
  - what the entity is currently doing
  - what the entity is capable of doing
  - who is controlling the entity
- Managing a wide variety of dead reckoning protocols is expensive
- Aircraft making military flight manoeuvers
  - constant acceleration and instant velocity ⇒ position trajectory
  - the aeroplane's orientation angle
- All information does not need to be transmitted
  - dancing is relevant not the footwork, fire not the flames,…
- In general, precise behaviour would be nice but overall behaviour is enough

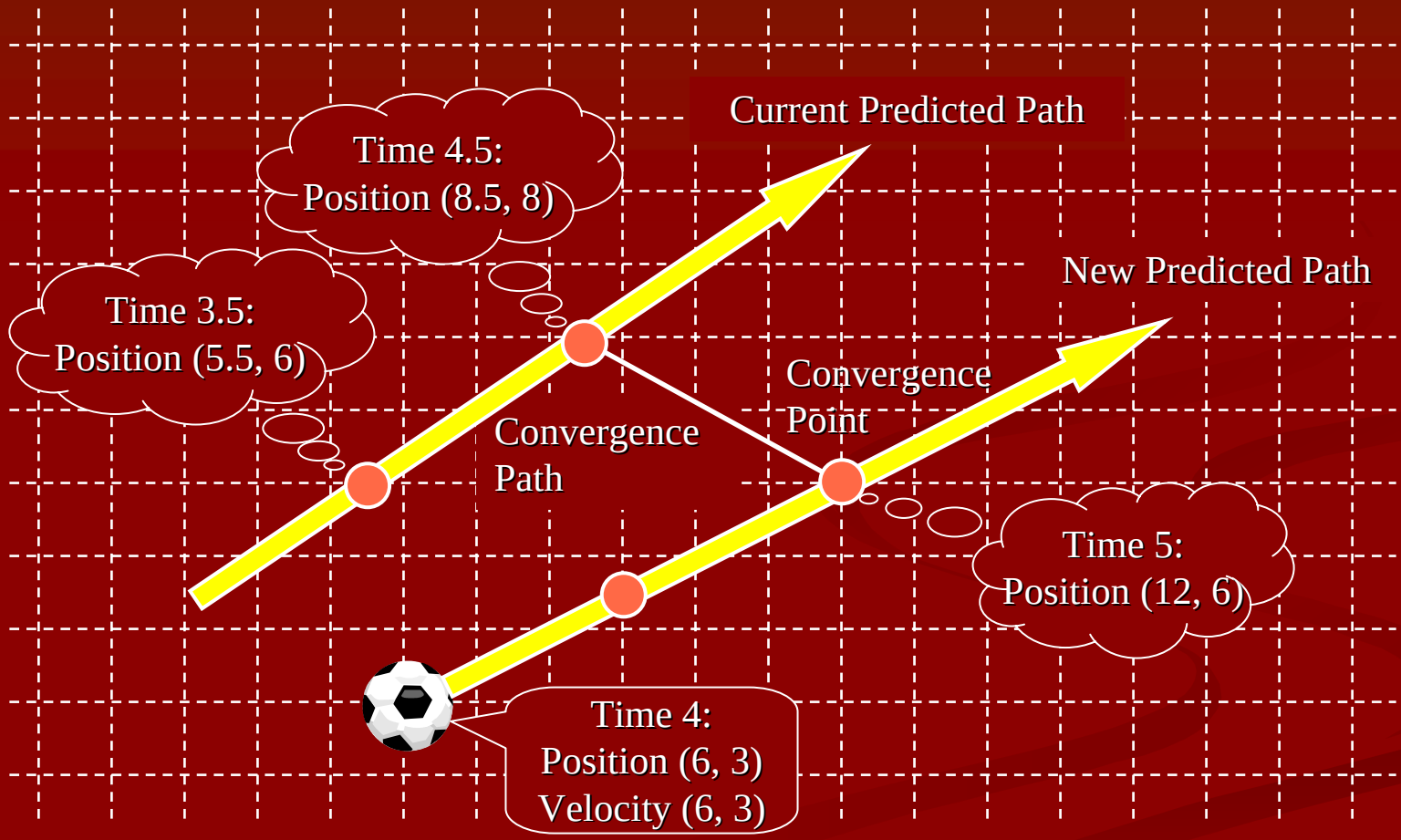
Convergence algorithms

- Prediction estimates the future value of the shared state
- Convergence tells how to correct inexact prediction
- Correct predicted state quickly but without noticeable visual distortion

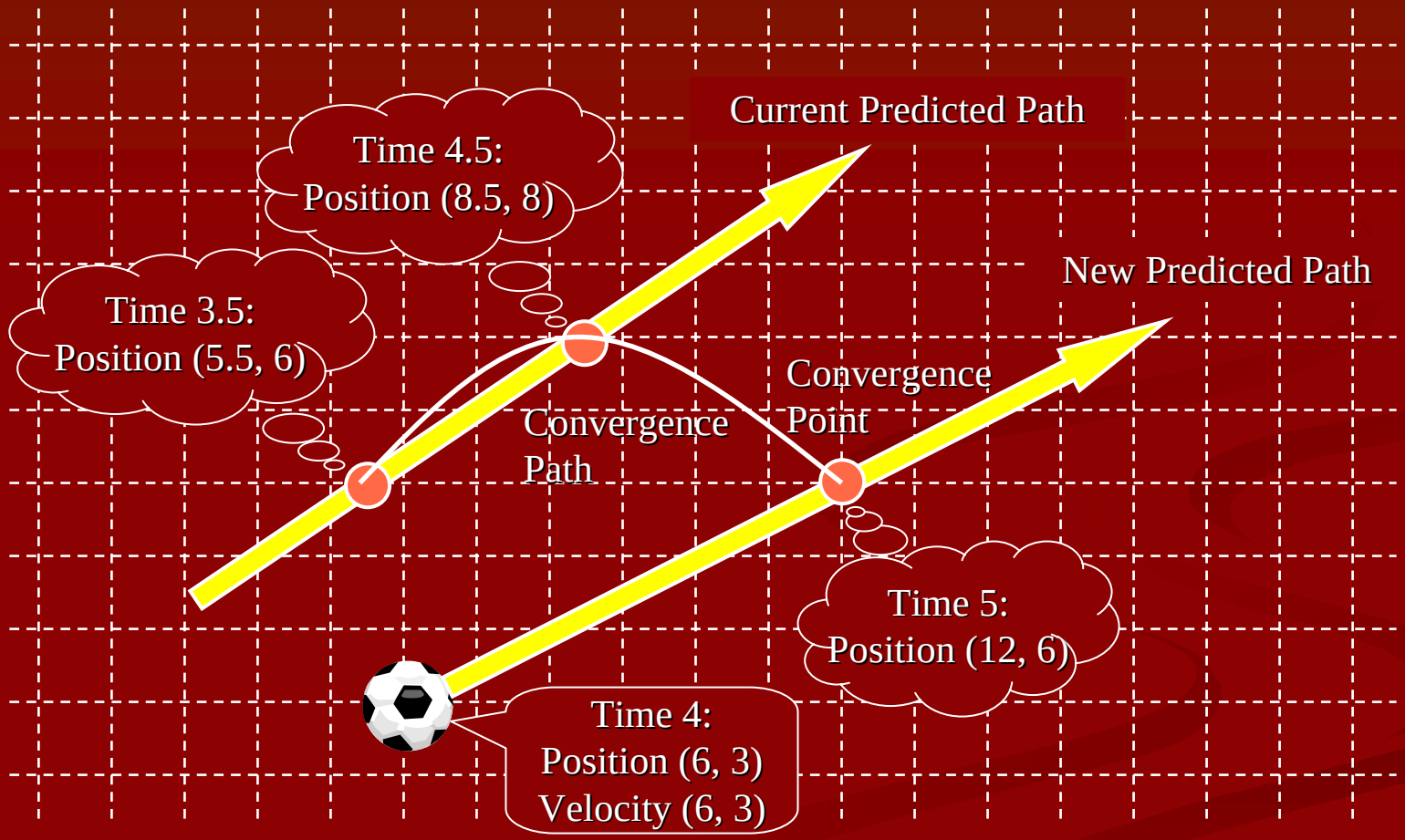
Zero-order convergence (or snap)



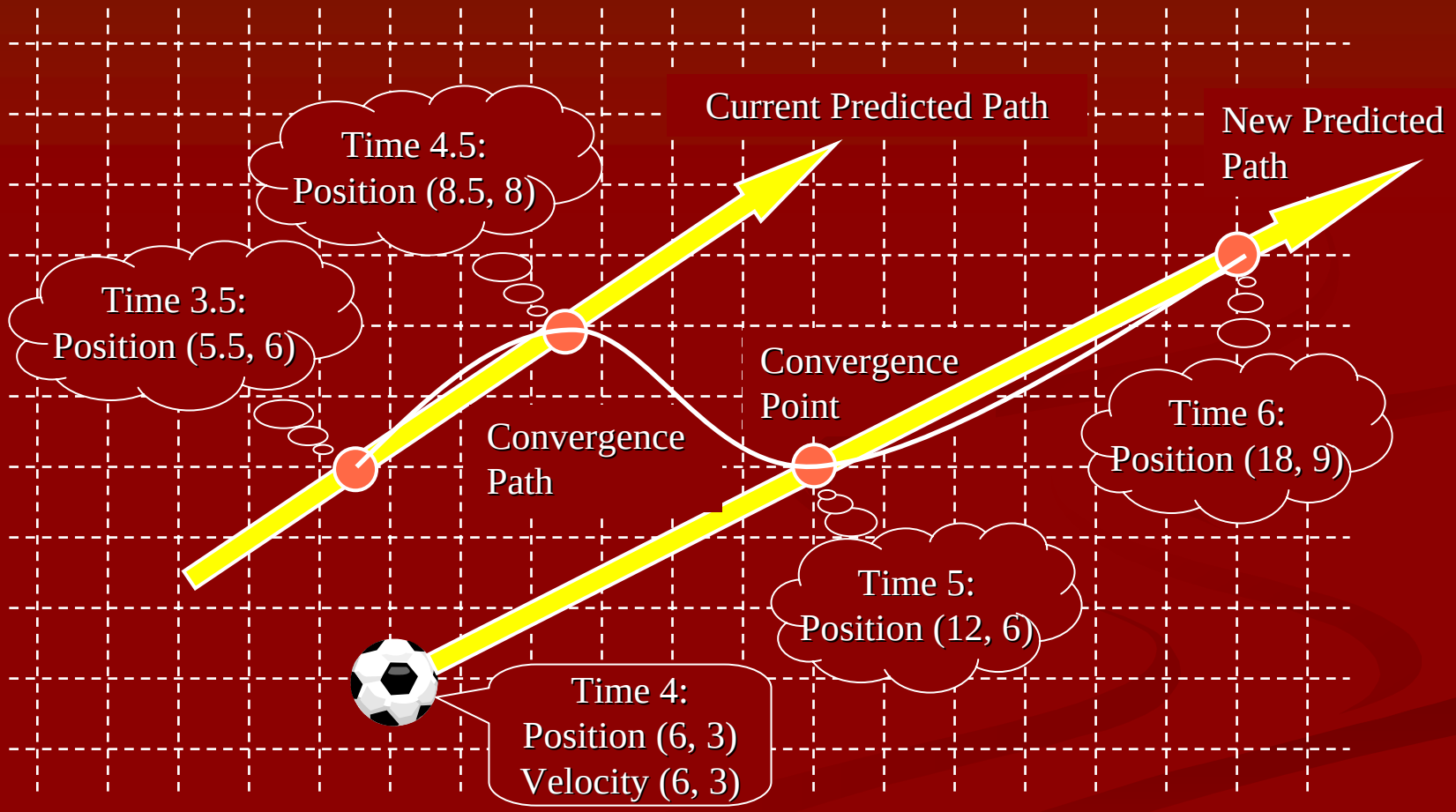
Linear convergence



Quadratic convergence



Convergence with cubic spline



Nonregular update generation

- By taking advance of knowledge about the computations at remote host, the source host can reduce the required state update rate
- The source host can use the same prediction algorithm than the remote hosts
- Transmit updates only when there is a significant divergence between the actual position and the predicted position

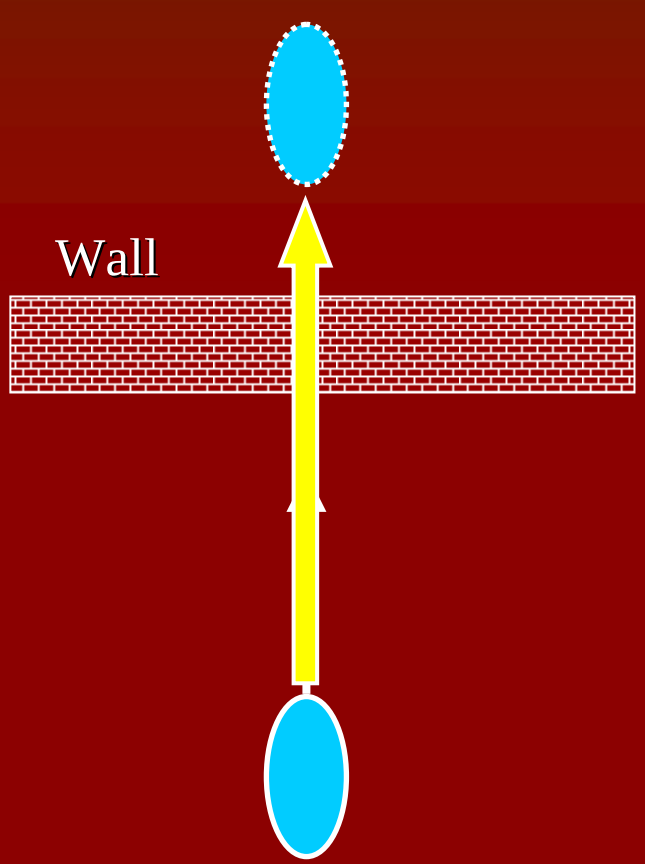
Advantages of nonregular transmissions

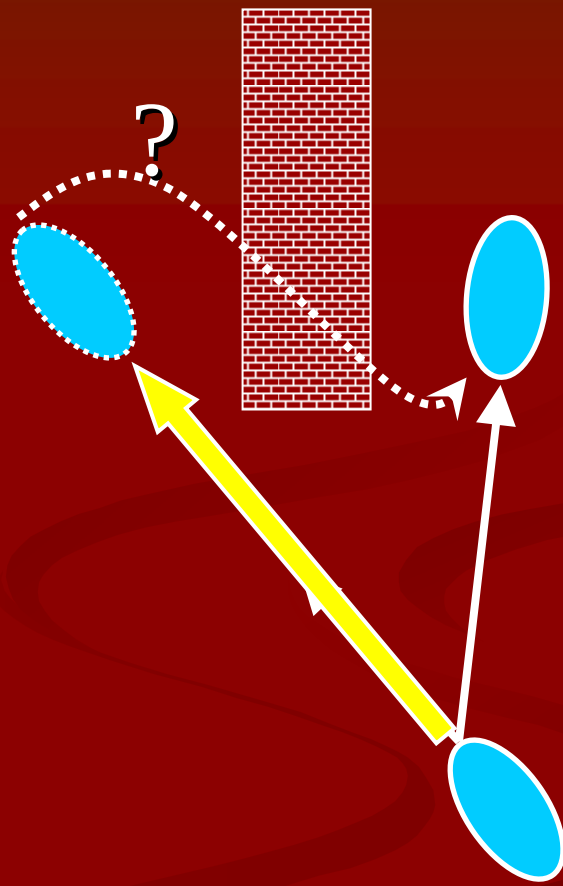
- Reduces update rates, if prediction algorithm is reasonable accurate
- Allows to make guarantees about the overall accuracy
- The source host can dynamically balance its network transmission resources
  - limited bandwidth ⇒ increase error threshold
- Nonregular updates provide a way to dynamically balance consistency and responsiveness based on the changing consistency demands

Lack of update packets

- If the prediction algorithm is really good, or if the entity is not moving significantly, the source might never send any updates
- New participants never receive any initial state
- Recipients cannot tell the difference between receiving no updates because
  - the object's behaviour has not changed
  - the network has failed
  - the object has left the game world
- Solution: timeout on packet transmissions

Environmental effects





Dead reckoning: advantages and drawbacks

- Reduces bandwidth requirements because updates can be transmitted at lower-than-frame-rate
- Because hosts receive updates about remote entities at a slower rate than local entities, receivers must use prediction and convergence to integrate remote and local entities
- Does not guarantee identical view for all participants
  - tolerate and adapt to potential differences
- Complex to develop, maintain, and evaluate
- Dead reckoning algorithms must often be customized for particular objects
- Are entities predictable?

Local perception filters

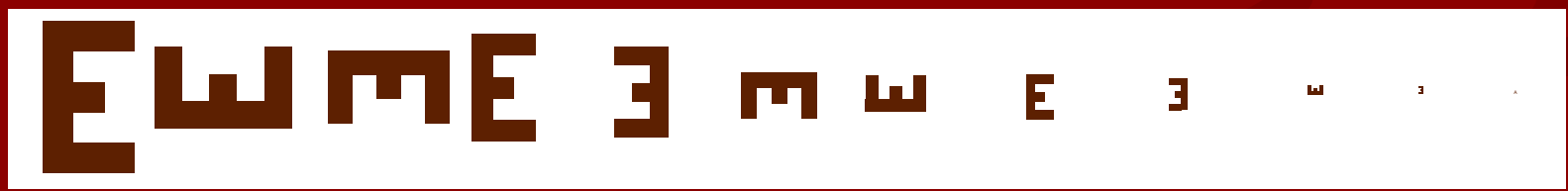
- exploiting human's perceptual limitations
  - level-of-detail: less details where they cannot be observed
  - image, video and audio compression
- local perception filters
  - exploits temporal perception
  - shows possibly out-of-date information (≠ dead reckoning)
  - ensures consistent interaction
  - allows to introduce artificial delays (e.g., bullet time)

Exploiting perceptual limitations

- Humans have inherent perceptual limitations
- Two approaches to exploit
  1. Information can provided at multiple levels of detail and at different update rates
  2. Mask the timeliness characteristics of information

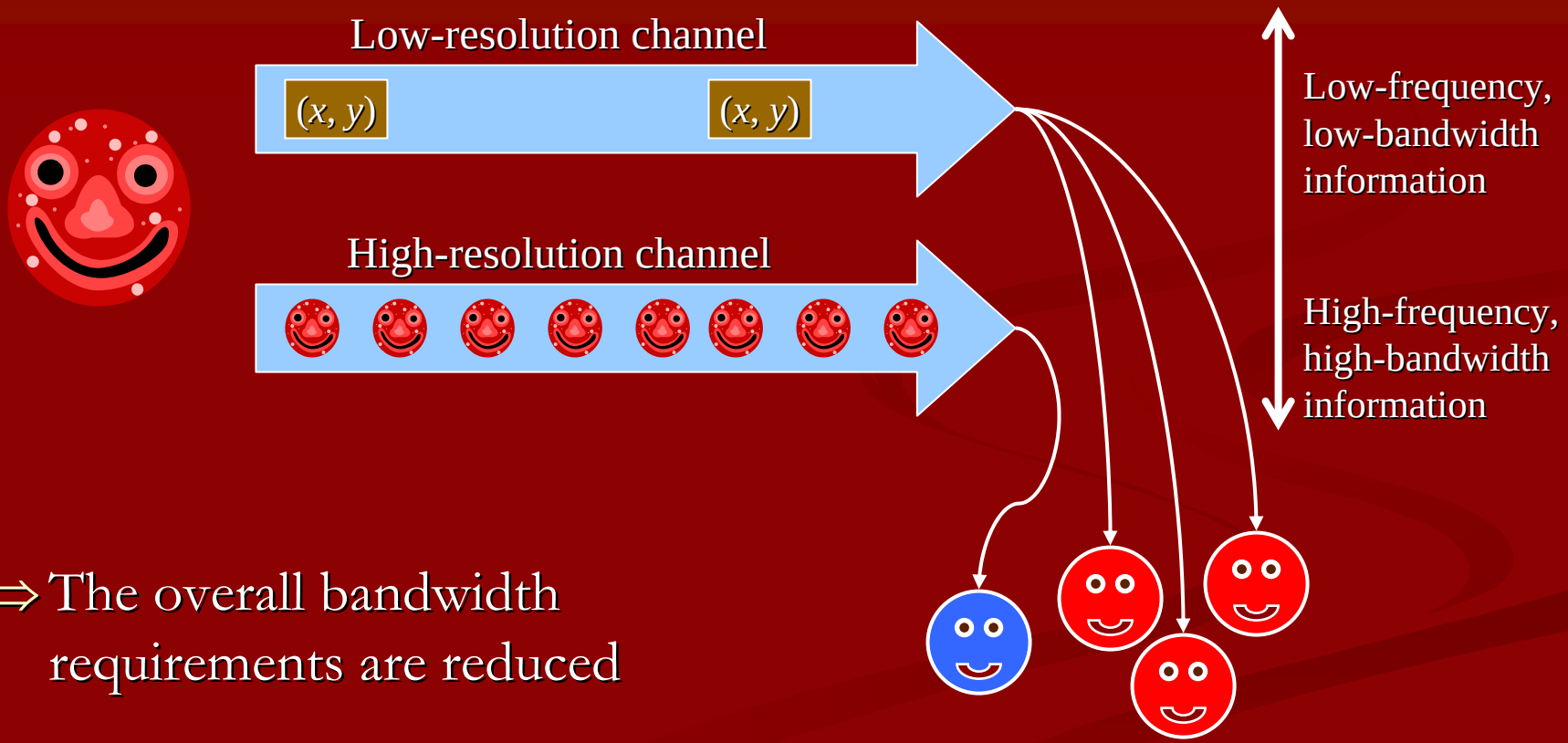
Exploiting level-of-detail perception

- Nearby viewers
  - expect full graphical details
  - accurate structure, position, orientation
  - update rate → local frame rate
- Distant viewers
  - can tolerate less graphical details
  - less accurate structure, position, orientation
- User's focus is typically nearby
- Many inaccuracies cannot even be detected on a fine-resolution display



Multiple-channel architecture

- Multiple independent data channels for each entity



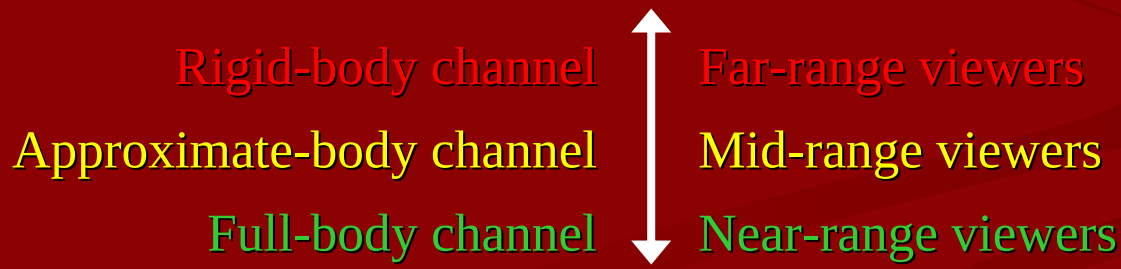
⇒ The overall bandwidth requirements are reduced

Implementation examples

- Client–server
  - each transmission identifies its channel
  - server dispatches data from channels to clients
- Multicast group for each region
  - assign multiple addresses for each region
    - one group provides all of the entities' high-resolution channels, another group provides all of the entities' low-resolution channels
- Multicast group for each entity
  - assign multiple addresses for each entity
- Different reliabilities to each channel
  - low-frequency updates are important
    - lost packets can have a significant impact

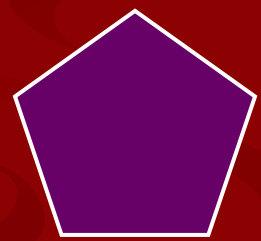
Selecting the channels to provide

- How many channels to provide for an entity?
  - more channels: better service for subscribers
  - each channel imposes a cost (bandwidth and computational)
- To satisfy the trade-off, three channels for each entity is typically needed
  - channels provide order-of-magnitude differences in
    - structural and positional accuracy
    - packet rate



Rigid-body channel

- Demands the least bandwidth and computation
- Represents the entity as a rigid body
- Ignores changes in the entity's structure
- Update types:
  - position
  - orientation
  - structure



Approximate-body channel

- More frequent position and orientation updates
- Hosts can render a rough approximation of the entity's dynamic structure
  - appendages and other articulated parts
- Provided information is entity-specific
  - corresponds to the dominant changes of the structure

Common approximations

- Radial length
  - motion towards and away from a centre point
  - update packets include the current radius
- Articulation vector
  - the current direction of the appendage
  - models a rotating turret, arms and legs
- Local co-ordinate system points
  - subset of the entity's significant vertices relative to the entity's local co-ordinate system
  - the entity is composed of multiple components

Full-body channel

- Highest level of detail
- High bandwidth and computational requirements
  - viewer can subscribe to a limited number of full-body channels
- Frequent transmissions
- Position and orientation
- Accurate structure information

Local perception filters (LPFs)

- introduced by Sharkey, Ryan & Roberts (1998)
- a method for hiding communication delays in networked virtual environments
- exploits the human perceptual limitations by rendering entities slightly out-of-date locations based on the underlying network delays
  - causality of events is preserved
  - rendered view may have temporal distortions
  - rendered view ≠ real view

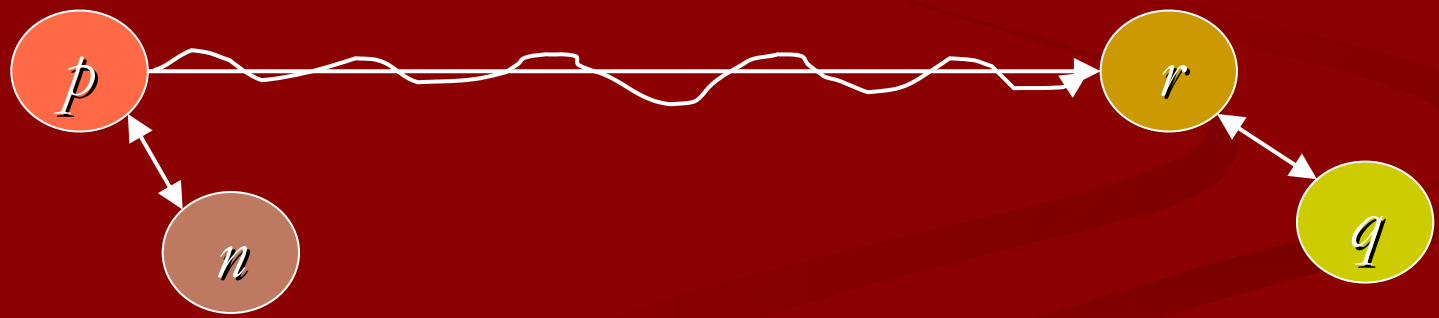
Active and passive entities

- An active entity (i.e., player)
  - takes actions on its own
  - generates updates
  - human participants, computer-controlled entities
  - cannot be predicted typically
  - rendered using state updates adjusted for the latency

- A passive entity
  - reacts to events from the environment, does not generate its own actions
  - inanimate objects (e.g., rocks, balls, books)
  - active entities interact with passive entities
  - rendered according to the latency of its nearest active entity
  - reacts instantaneously to the actions of a nearby active entity

Rules of LPFs

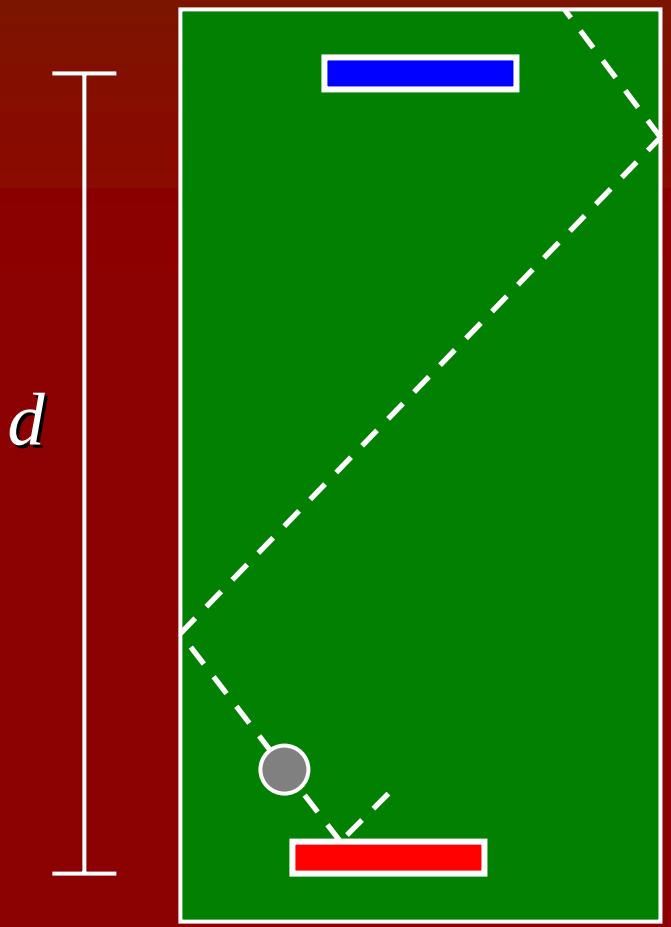
1. Player should be able to interact in real-time with the nearby entities.
2. Player should be able to view remote interactions in real-time, although they can be out-of-date.
3. Temporal distortions in the player's perception should be as unnoticeable as possible.



Interaction between players

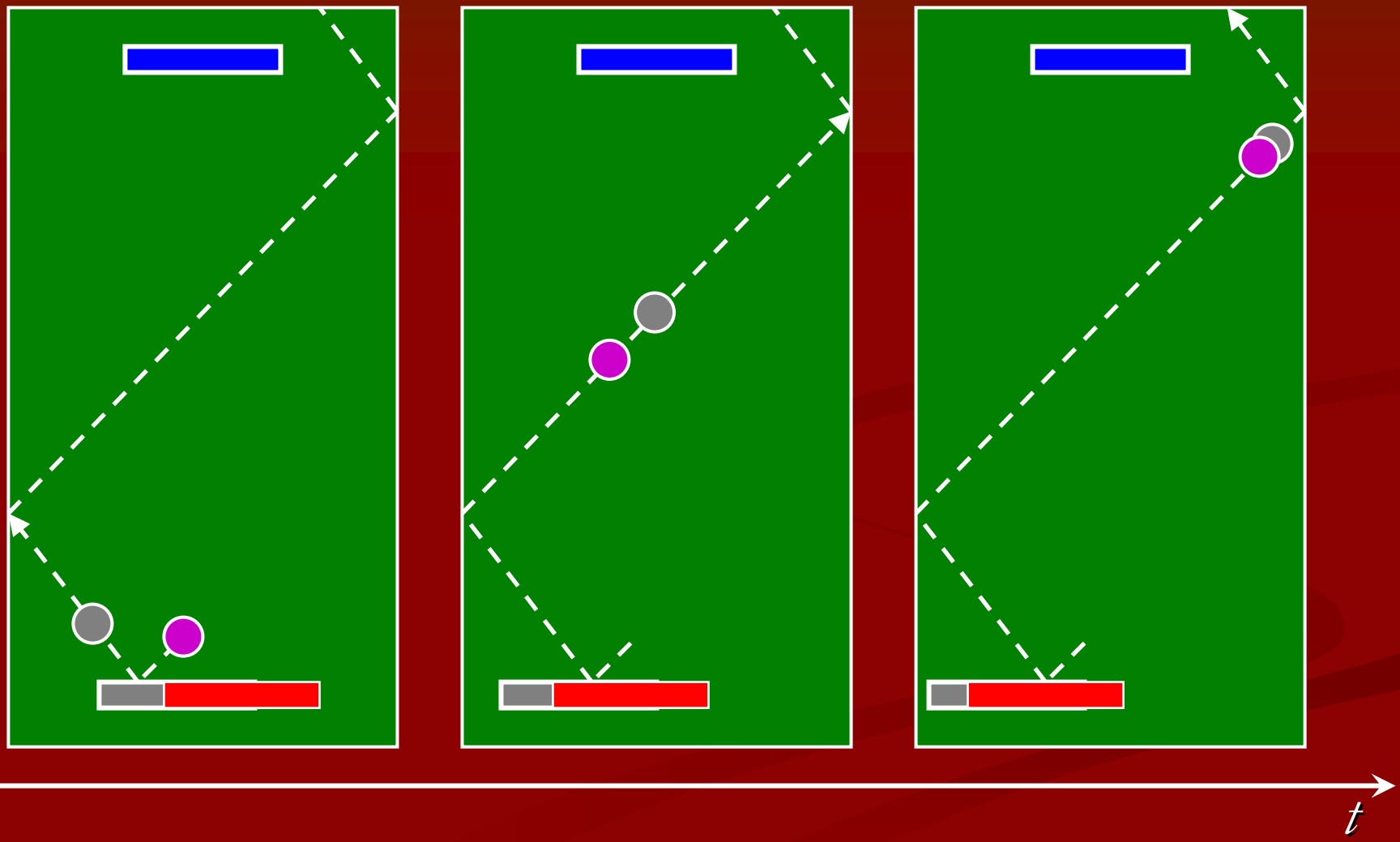
- interaction = communication between the players
  - local players: immediate
  - remote players: subject to the network latency
    - time frame = current time – communication delay
- interaction = players exchanging passive entities
  - passive entities are predictable ⇒ they can be rendered in the past (or in the future)
- a passive entity can change its time frame dynamically
  - the nearer to a local player, the closer it is rendered to the current time
  - the nearer to a remote player, the closer it is rendered to its time frame

Example: Pong

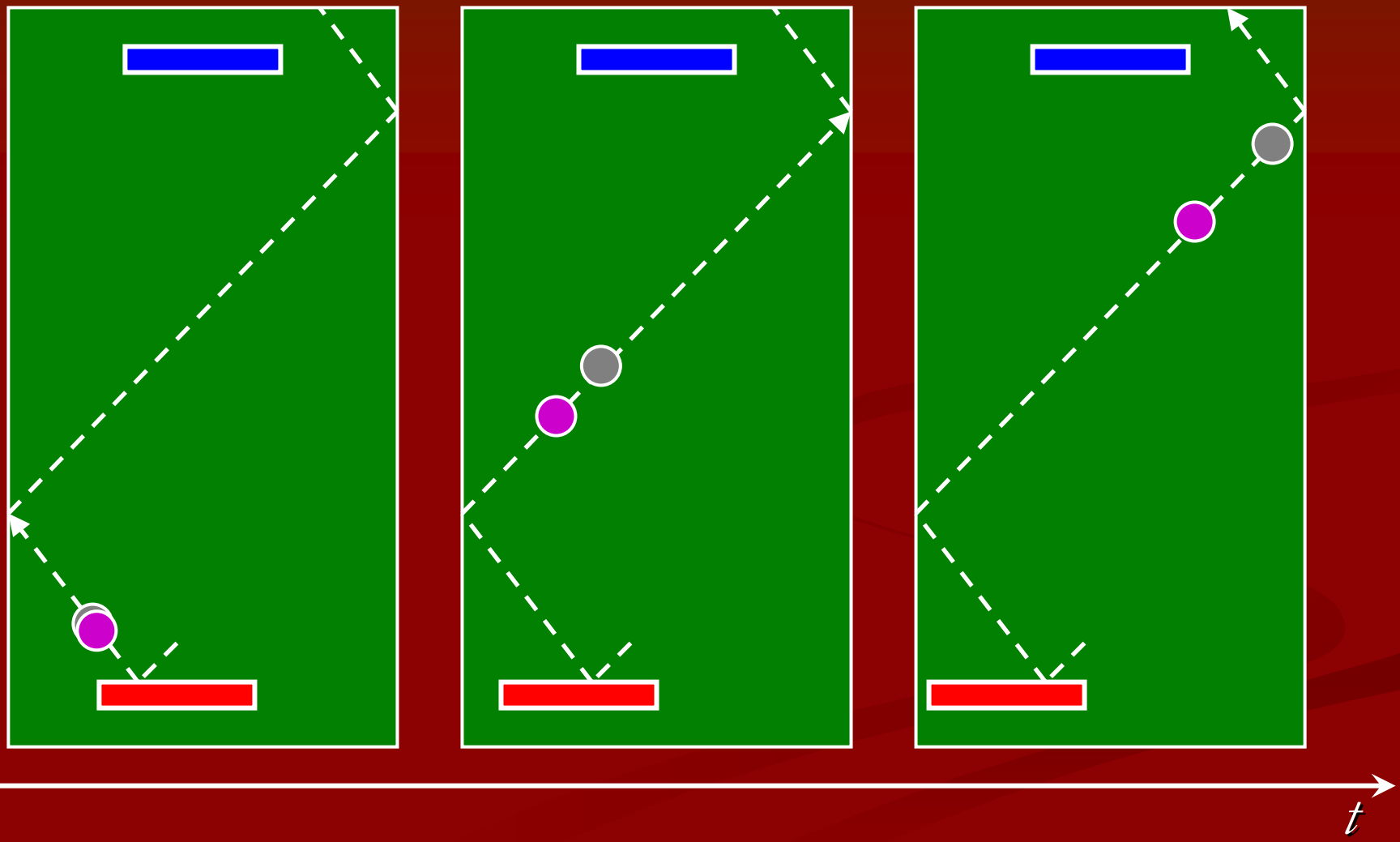


- Two active entities: paddles
  - movement unpredictable
- One passive entity: ball
  - movement predictable
- Latency of *d* seconds

The view of the blue player



The view of the red player

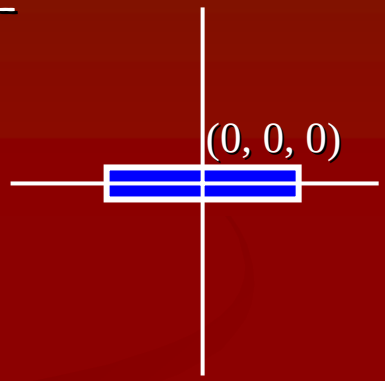


Pong: summary

- Each player sees a different representation of the same playing field
- The ball accelerates as it approaches the local player's paddle
- The ball decelerates as it approaches the remote player's paddle
- The ball's rendered position alternates between
  - the current time
    - meaningful interaction for local player
  - a past time reference
    - network latency
    - observing meaningful interaction for remote player

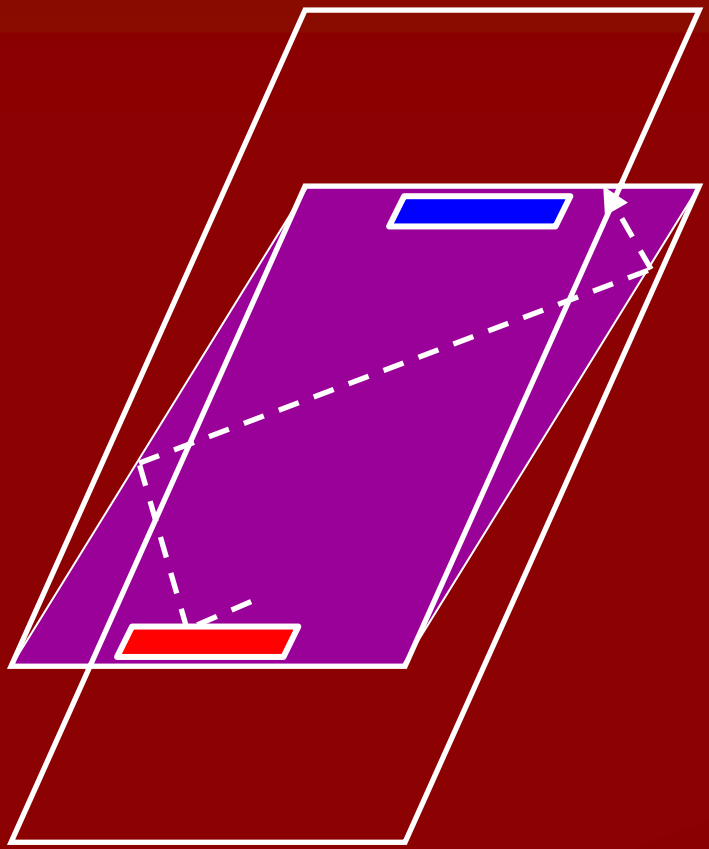
3½-dimensional temporal contour

- Represent each player's perception as a four-dimensional co-ordinate system (x, y, z, t)
  - x, y, z: the spatial position relative to the local player's current position
    - local player at (0, 0, 0)
  - t: the time associated with rendered information from that position
    - local player rendered at current time: $t = 0$
    - opposing player: $t = -d$

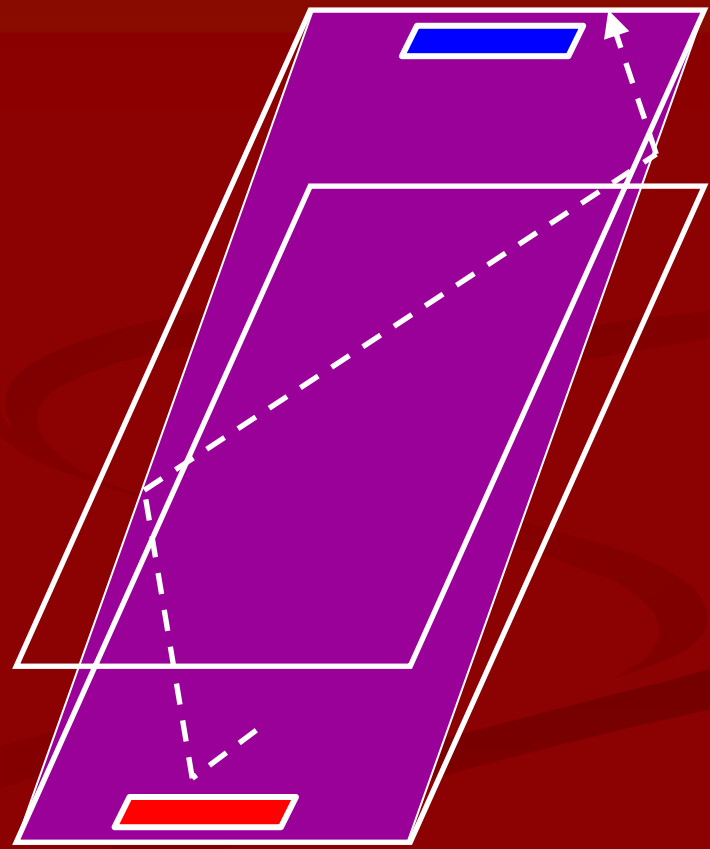


Temporal contours in Pong

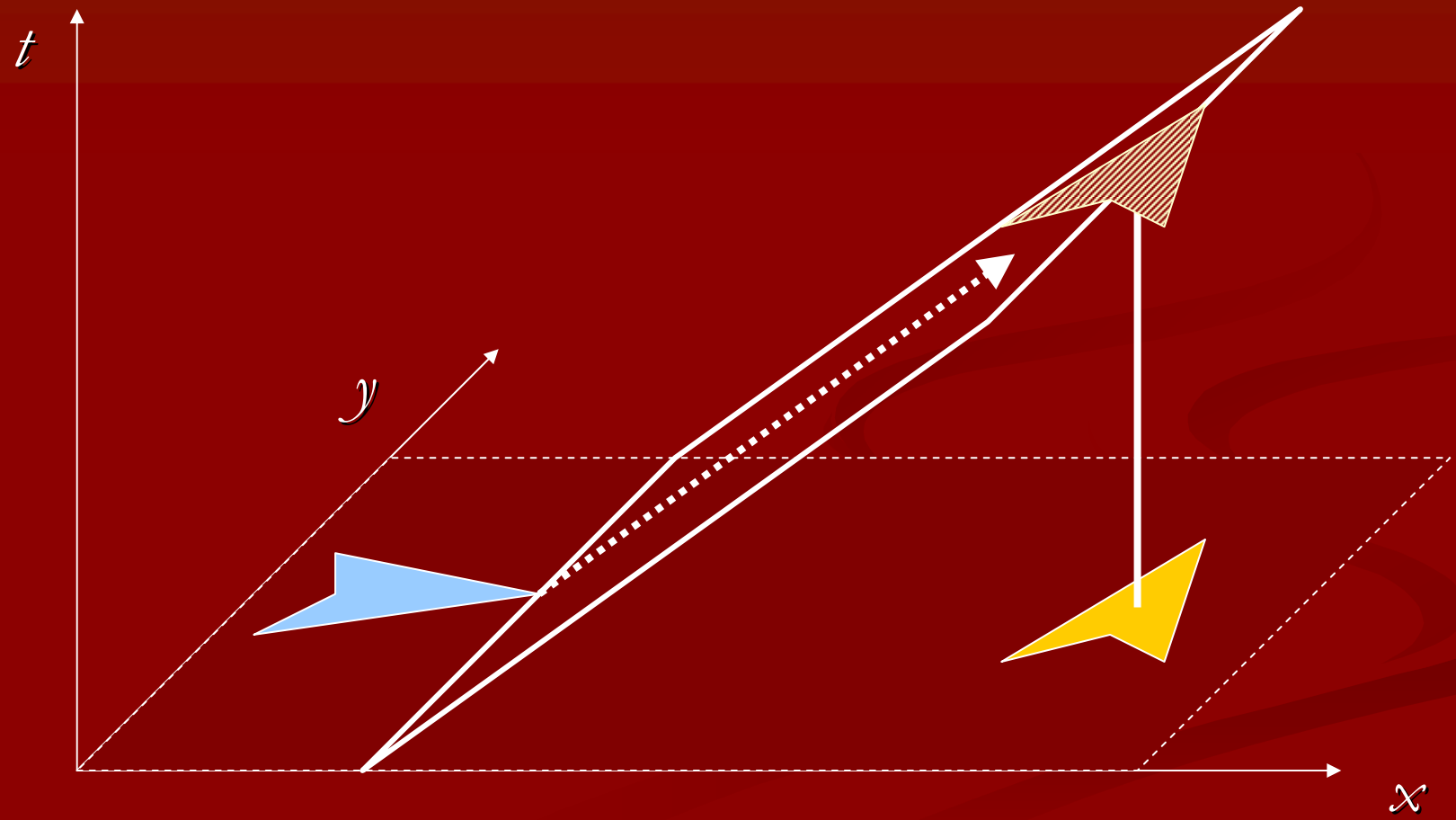
Blue player



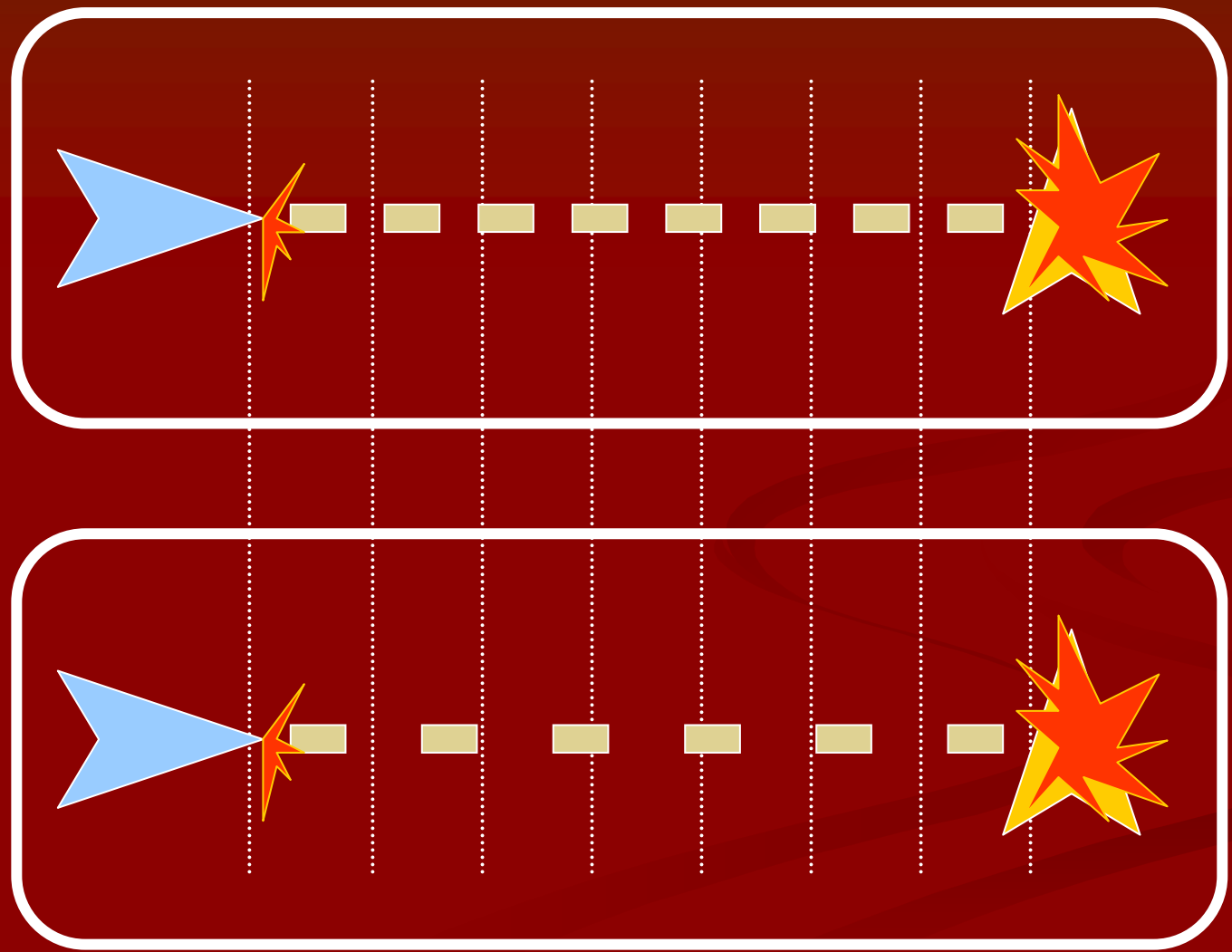
Red player



Temporal contour (from the blue player's perspective)

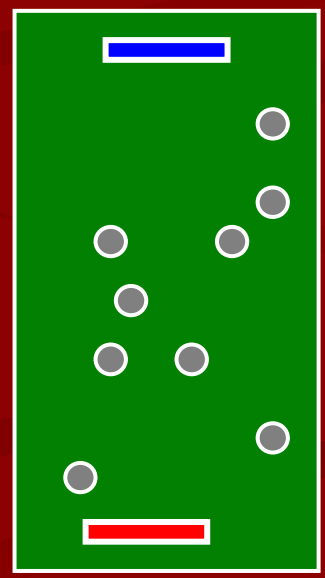


Temporal distortion



Properties of the co-ordinate system

- The co-ordinate system is defined independently for each player
- Depends on the player's current position and the delay of arriving information
- Changes dynamically as the player moves or as the network properties change
- Defines how a passive object should be rendered
- Two interacting objects are rendered at the same time reference point
- Each user perceives all collisions correctly
- Objects that approach the local user are rendered in the user's time
- Smooth movement

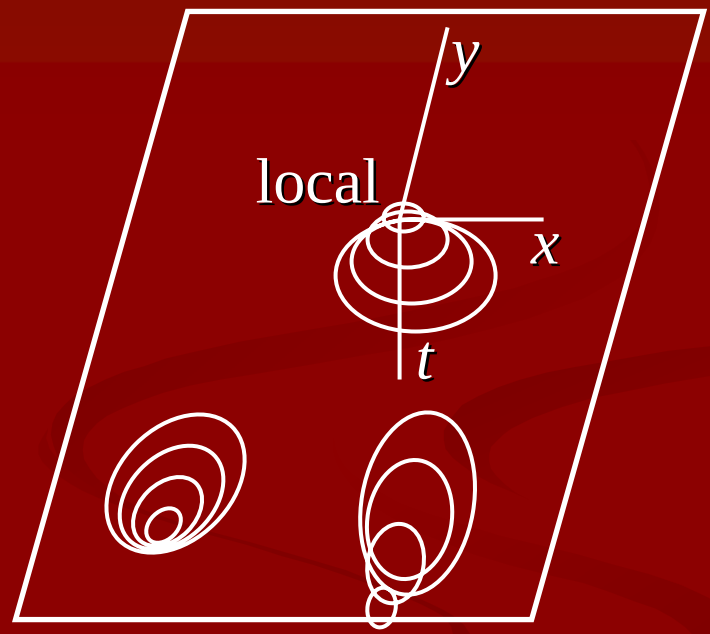


Generalizing the local temporal contour

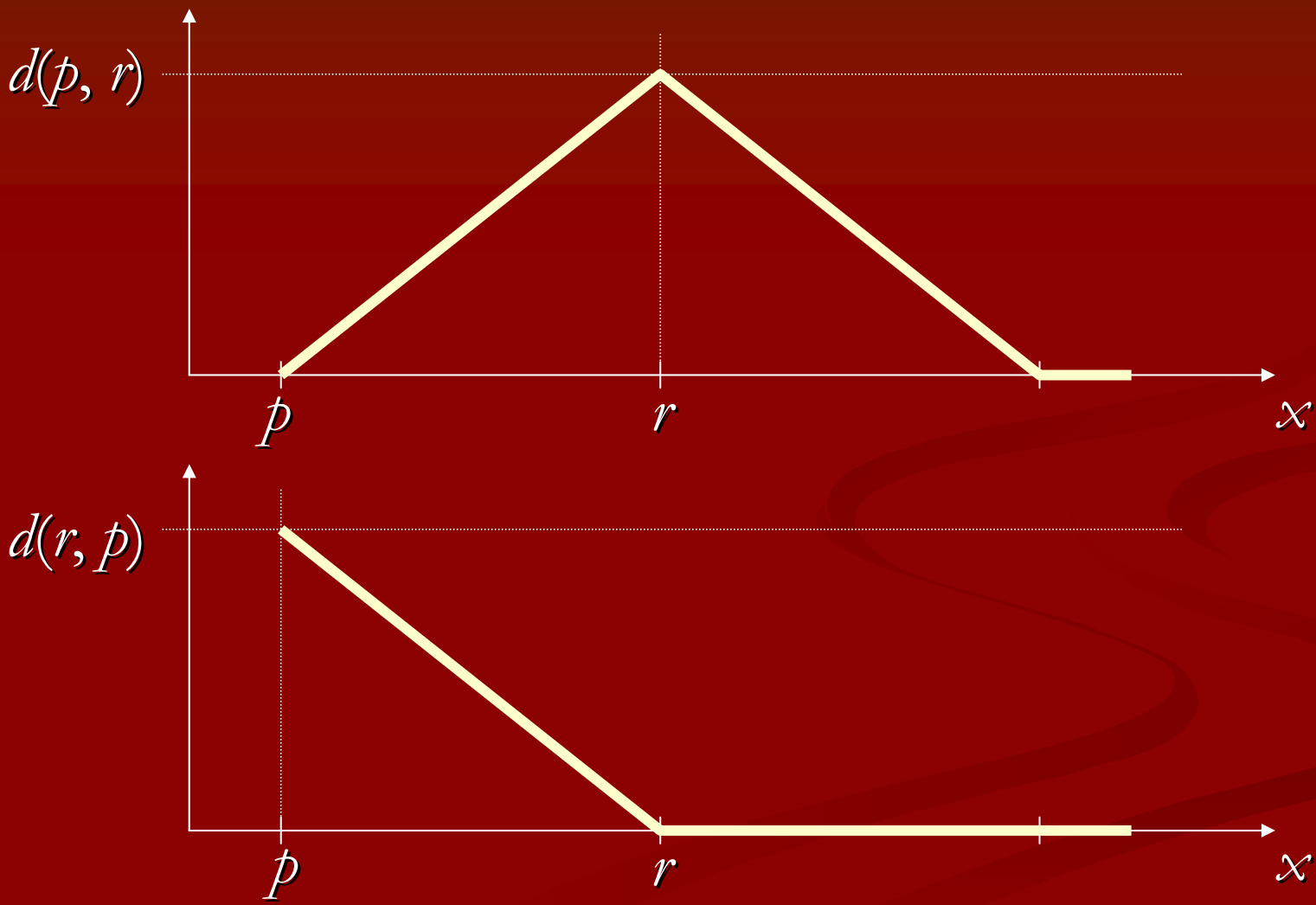
- Limitations:
  - players are capable of moving along a single axis only
  - supports two active objects only
- Generalization to a 4D co-ordinate system requires preserving for the local user:
  - interacting naturally with passive objects in vicinity
  - seeing remote interactions (passive-to-passive, passive-to-active) naturally
  - perceiving smooth motion of remote objects

Local temporal contour

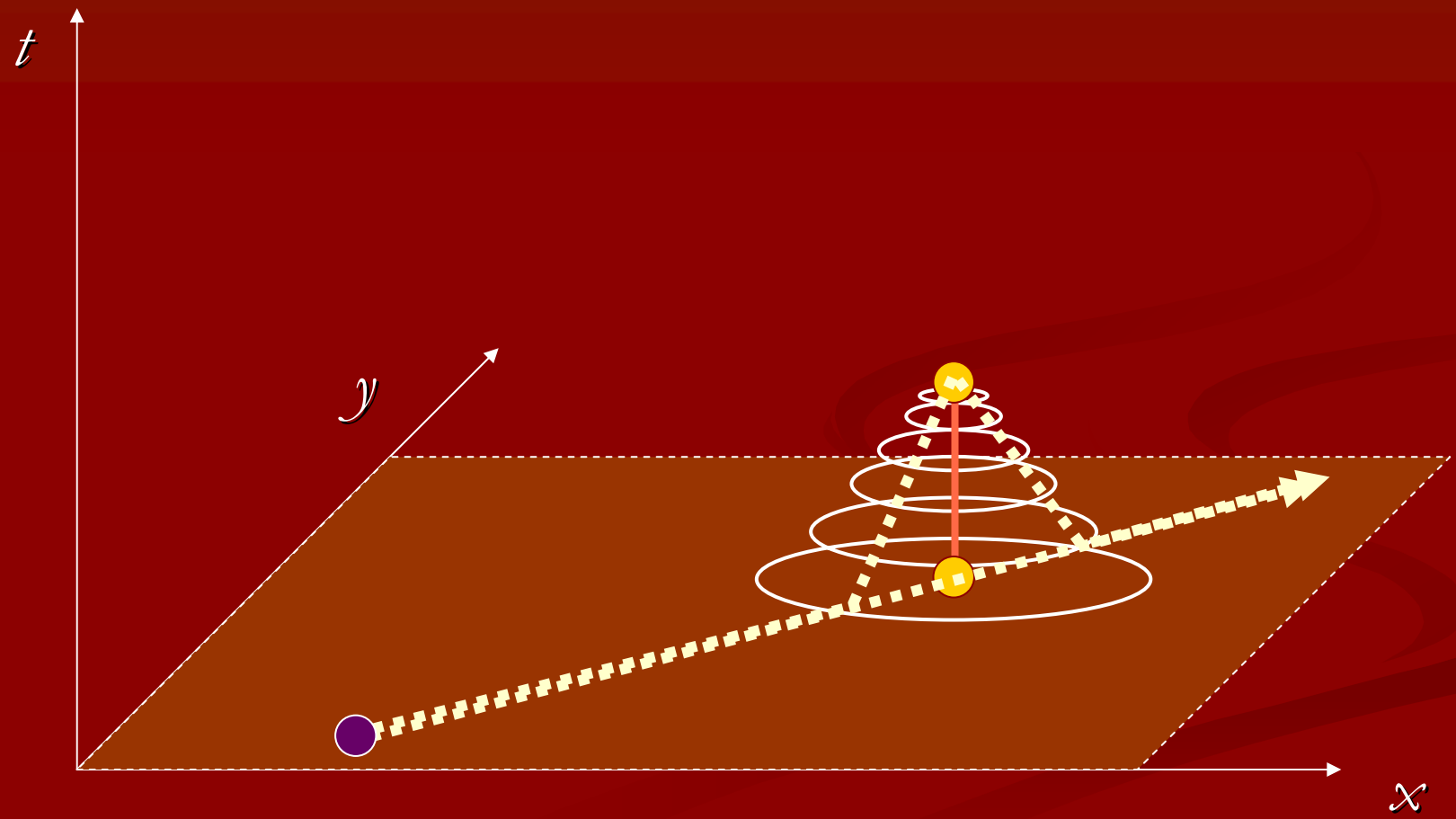
- The local user at (0, 0, 0)
- Each active object is assigned a *t* value corresponding to its latency
- Interpolate the contour over all active objects including local
- Contour defines a suitable *t* value for each spatial point



Linear temporal contours



2½-dimensional temporal contour



Multiple players: aggregating the temporal contours

Worth noting

- simple linear functions instead of continuous temporal contours
- LPFs are the 'opposite' of dead reckoning
  - no prediction for remote players
- the closer the players get, the more noticeable the temporal distortion becomes
  - in critical proximity interaction becomes impossible
  - no mêlée

Problems

- possibly visual disruptions on impact ⇒ shadows (see the lecture notes for details)
- sudden changes in the player's position or delay can cause unwanted effects
  - if a player leaves the game, what happens to the temporal contour?
  - third party instrusion: someone with a high delay 'blocks' the incoming entities
  - jitter: entities start to bounce back and forth in time

Bullet time

- movies: visual effect combining slow motion with dynamic camera movement
- computer games: player can slow down the surroundings to have *more time* to make decisions
- easy in single player games: slow down the game!
- how about multiplayer games?

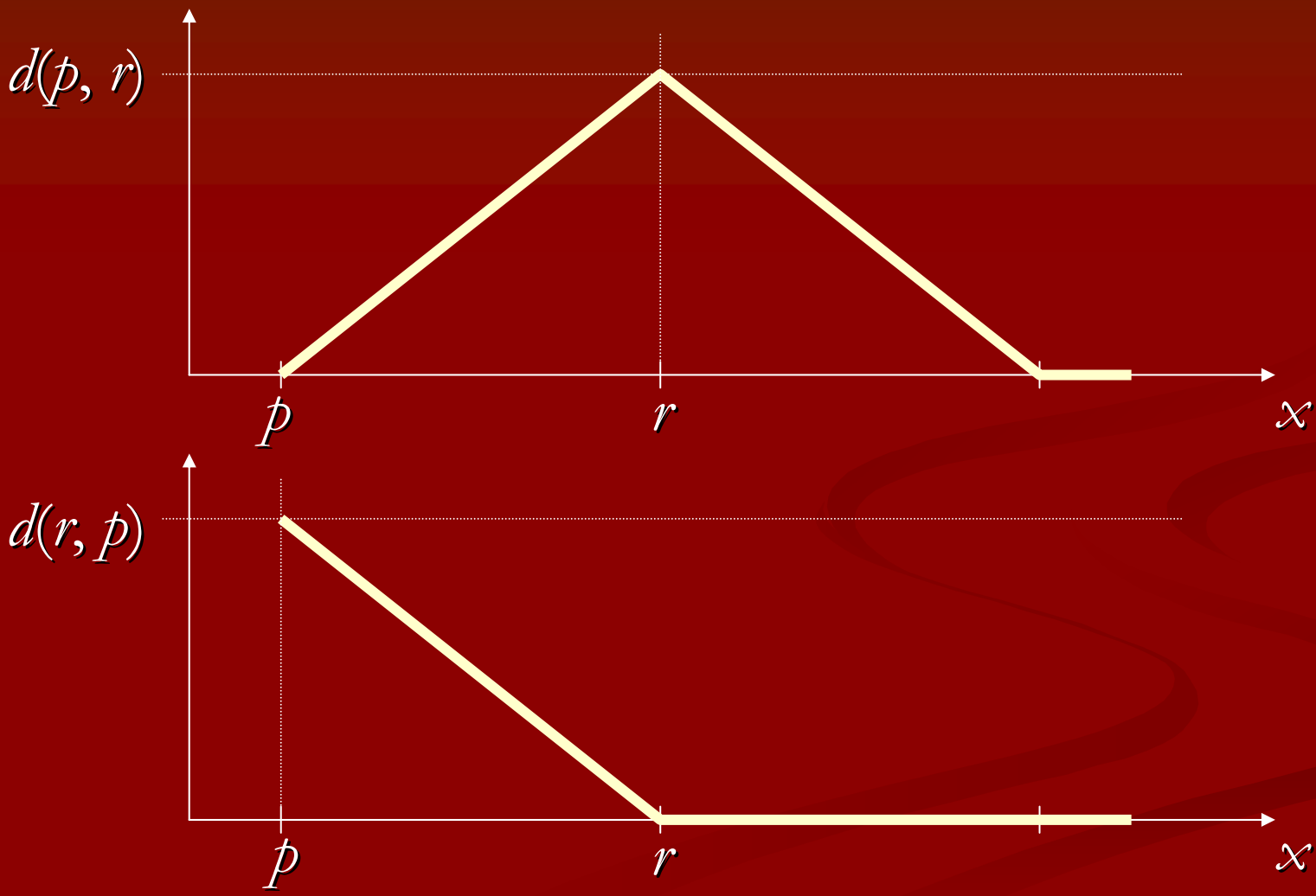
Bullet time in multiplayer games

- two approaches:
  - speed up the player
  - slow down the other players
- if a player can slow down/speed up the time, how it will affect the other players?
  - localize the temporal distortion to the immediate surroundings of the player
- but how to do that?
- $\Rightarrow$ local perception filters!

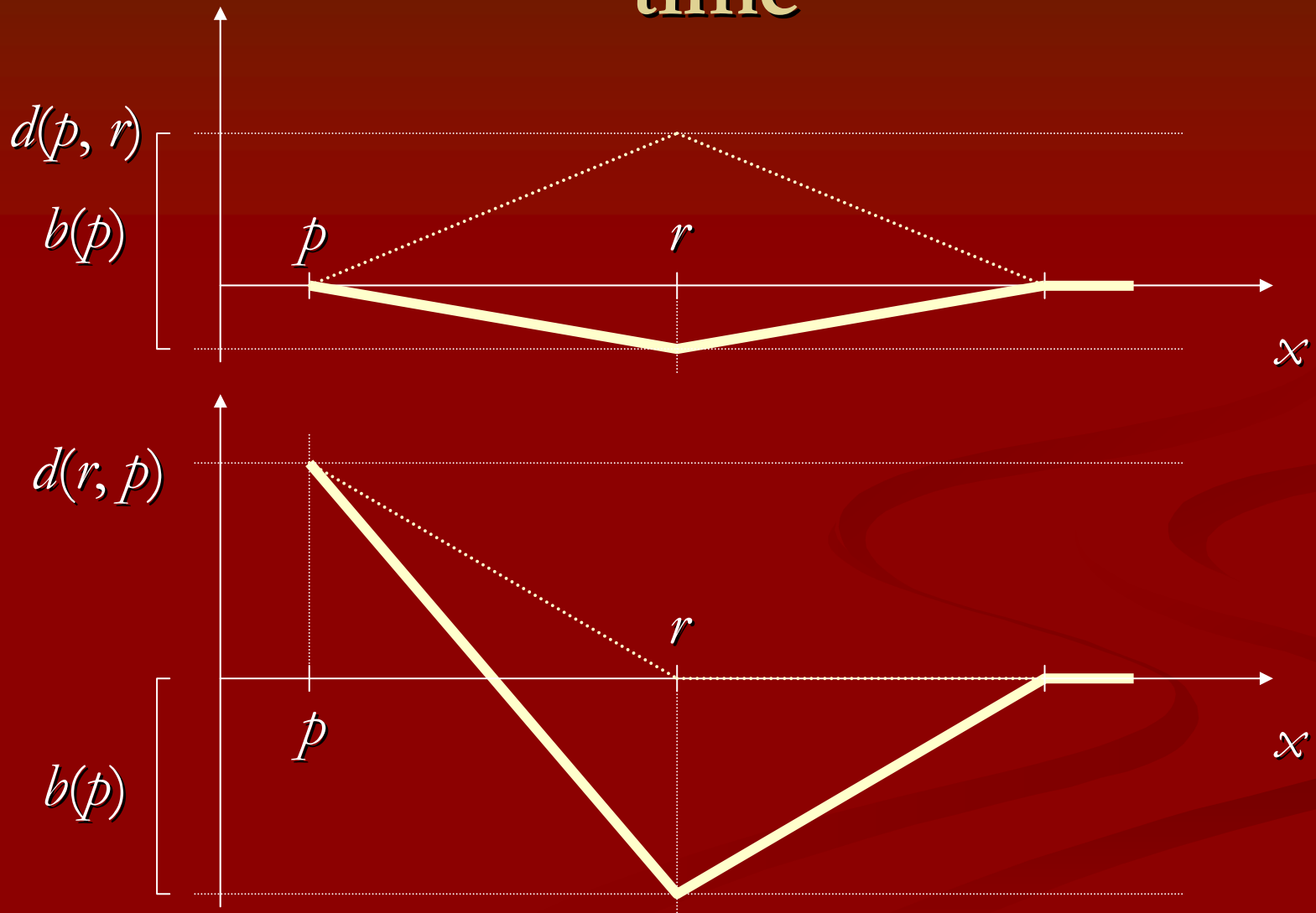
Adding bullet time to LPFs

- player using the bullet time has more time to react
  $\Rightarrow$ the delay between bullet-timed player and the other players increases
- add artificial delay to the temporal contour

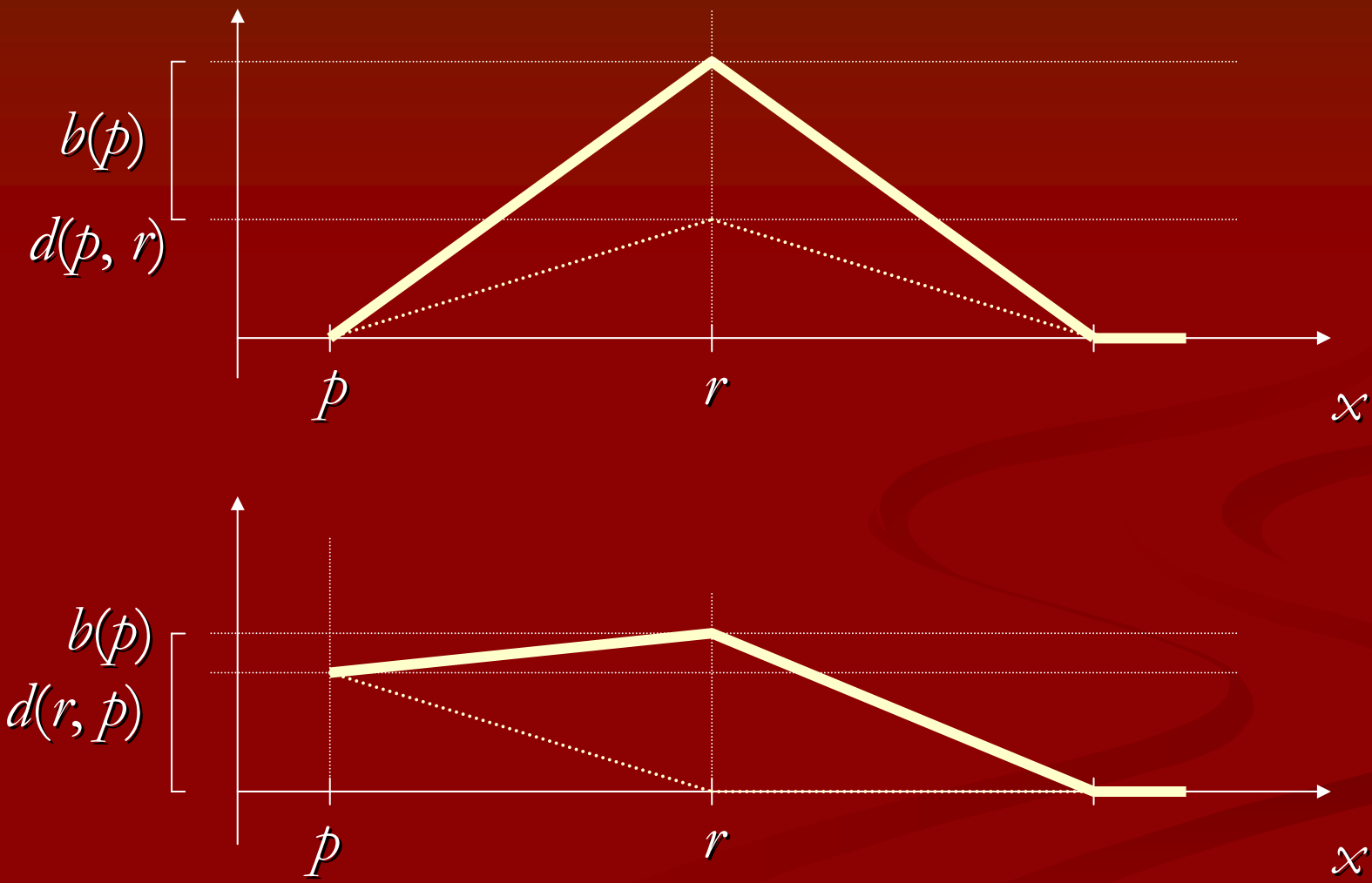
p shoots *r* without bullet time



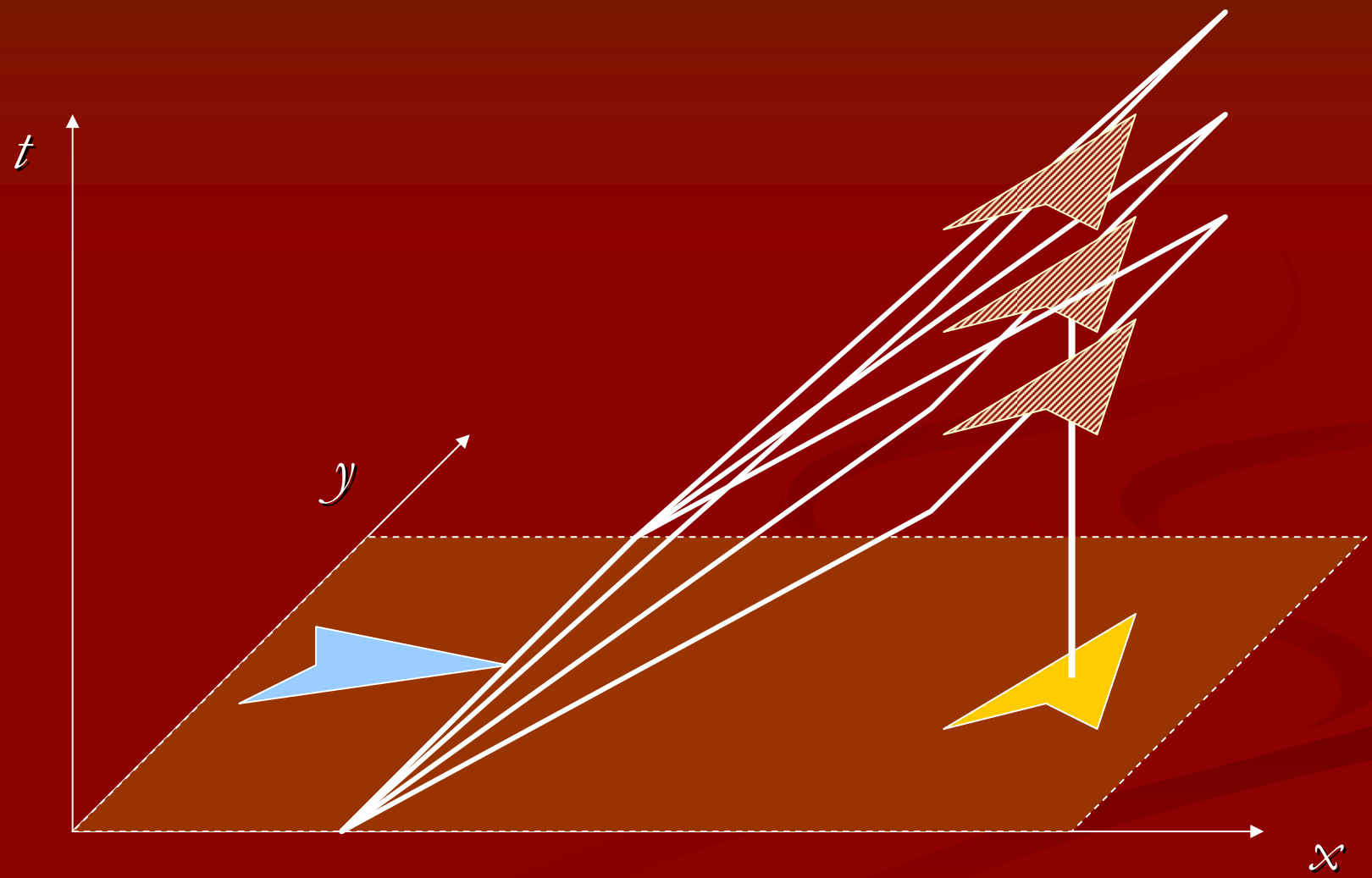
p shoots r while p is using bullet time



p shoots *r* while *r* is using bullet time



2½-dimensional temporal contour and bullet time



Open questions

- non-linear temporal contours
  - how to compute quickly?
  - noticeable benefits (if any)?
- numerical evaluation
  - measuring the distortion and its effects
- practical evaluation
  - how well does it work?
  - does it allow new kinds of games?

Synchronized simulation

- used in *Age of Empires* (1997)
- command categories:
  - deterministic: computer
  - indeterministic: human
- distribute the indeterministic commands only
- deterministic commands are derived from pseudo-random numbers
  → distribute the seed value only
- consistency checks and recovery mechanisms

Synchronized simulation in *Age of Empires*

- *Age of Empires* game series by Ensemble Studios
- Real-time strategy (RTS) game
- Max 8 players, each can have up to 200 moving units
  $\Rightarrow$ 1600 moving units
  $\Rightarrow$ large-scale simulation
- Rough breakdown of the processing tasks:
  - 30% graphic rendering
  - 30% AI and path-finding
  - 30% running the simulation and maintenance

Synchronized (or simultaneous) simulation

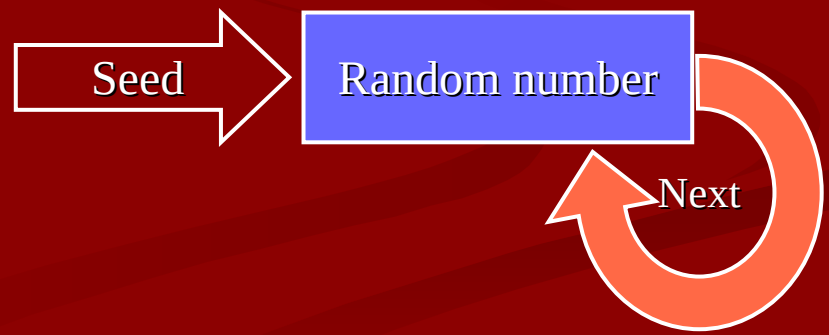
- Large simulation ⇒ a lot of data to be transmitted
- Trade-off: computation vs. communication
  - 'If you have more updating data than you can move on the network, the only real option is to generate the data on each client'
- Run the *exact* same simulation in each client

Handling indeterminism

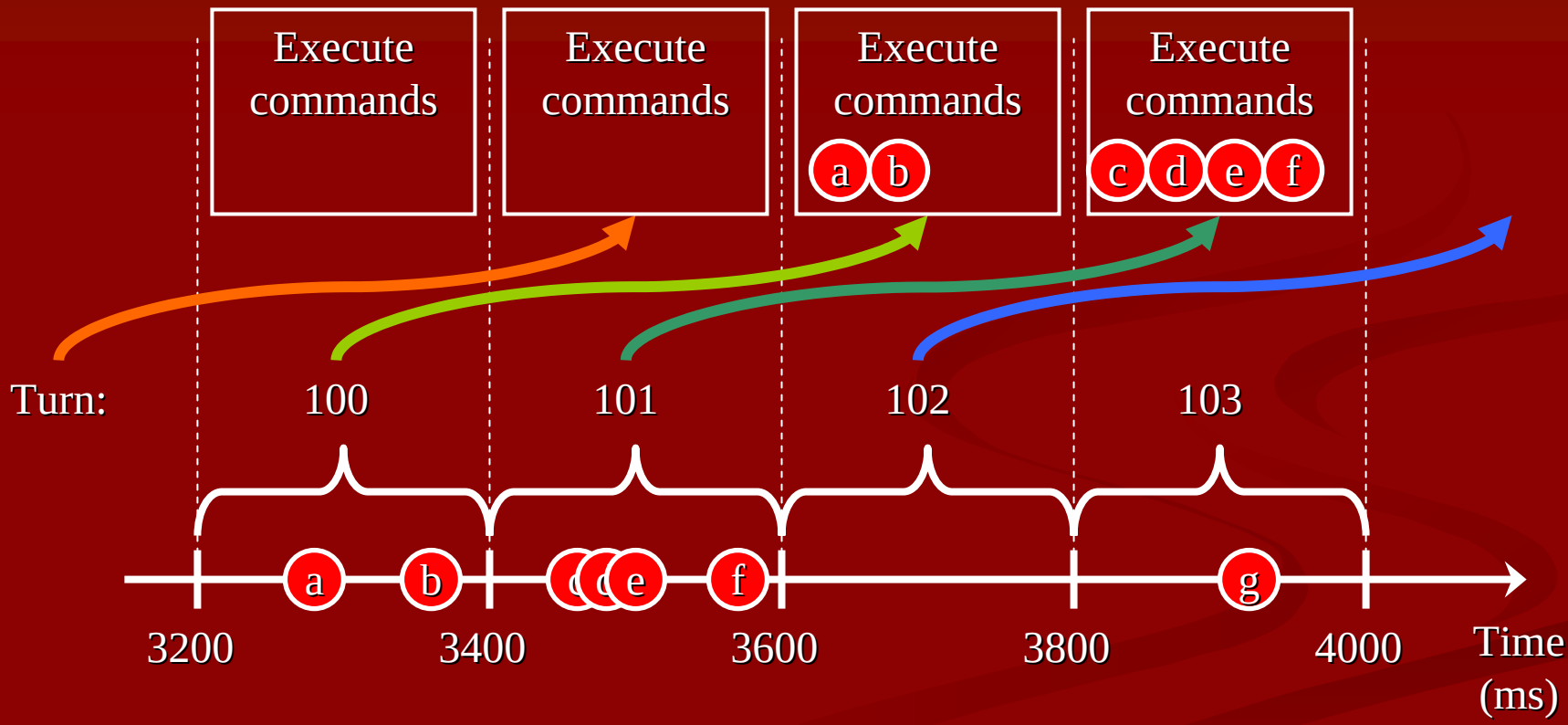
- 'Indeterministic' events are either
  - predictable (computers) or
  - unpredictable (humans)
- Only the unpredictable events have to be transmitted ⇒ communication
  - apply an identical set of commands that were issued at the same time
- The predictable events can be calculated locally on each client ⇒ computation

- Pseudo-random numbers are deterministic
- All clients use the same seed for their random number generator
  - disseminate the seed

Pseudo-random number generator



Communication turns



Features

- Guaranteed delivery using UDP
  - message packet:
    - execution turn
    - sequence number
  - if messages are received out of order, send immediately a resend request
  - if acknowledgement arrives late, resend the message
- Hidden benefits
  - clients are hard to hack
  - any simulation running differently is out-of-sync
- Hidden problems
  - programming is demanding
  - out-of-sync errors
  - checksums for everything
    - 50 Gb message logs

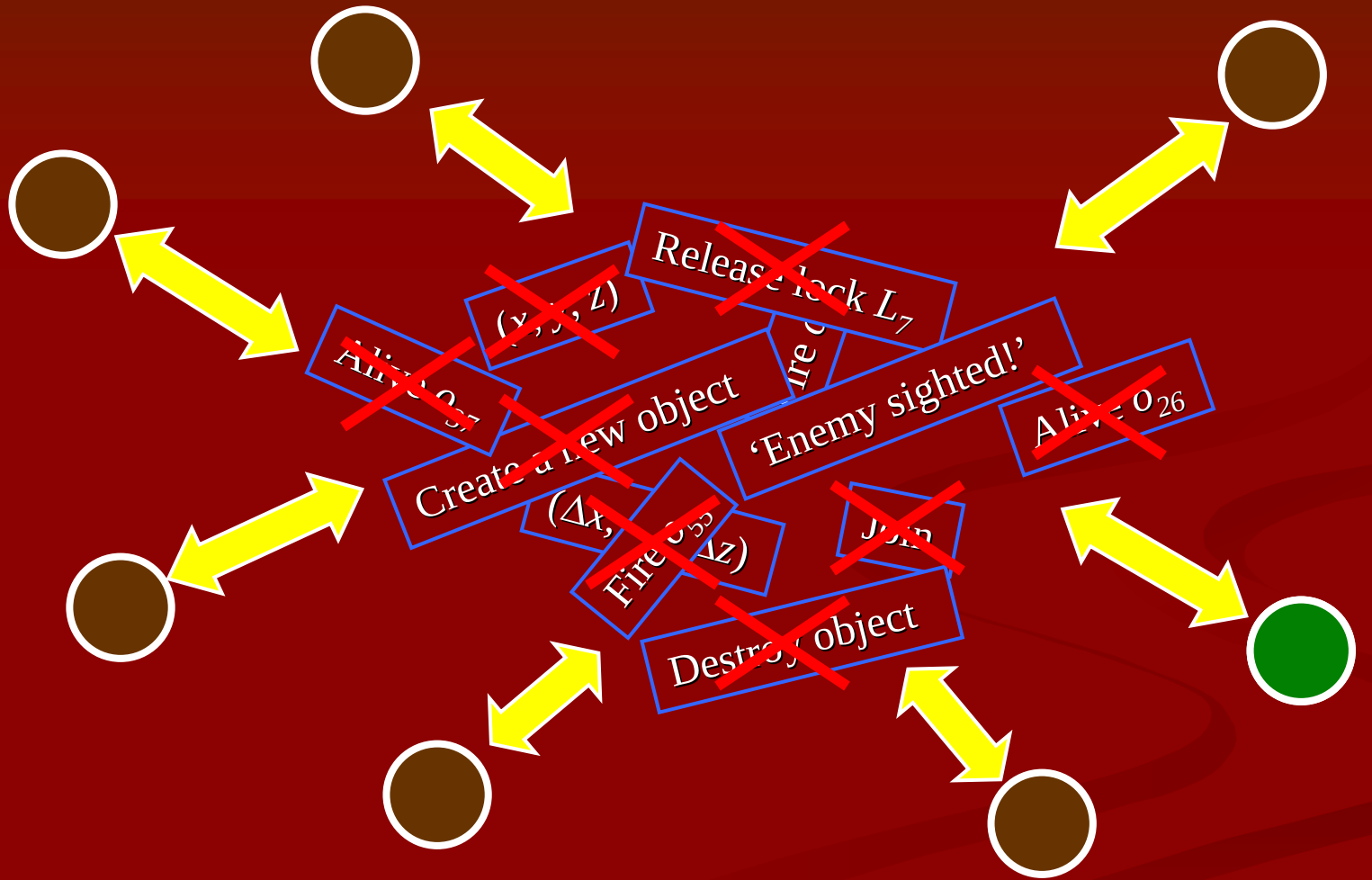
Lessons learned

- Players can tolerate a high latency as long as it remains constant
  - for an RTS game, even 250–500 ms latencies are still playable
- Jitter (the variance of the latency) is a bigger problem
  - consistent slow response is better than alternating between fast and slow
- Studying player behaviour helps to identify problematic situations
  - hectic situations (like battles) cause spikes in the network traffic
- Measuring the communication system early on helps the development
  - identify bottlenecks and slowdowns
- Educating programmers to work on multiplayer environments

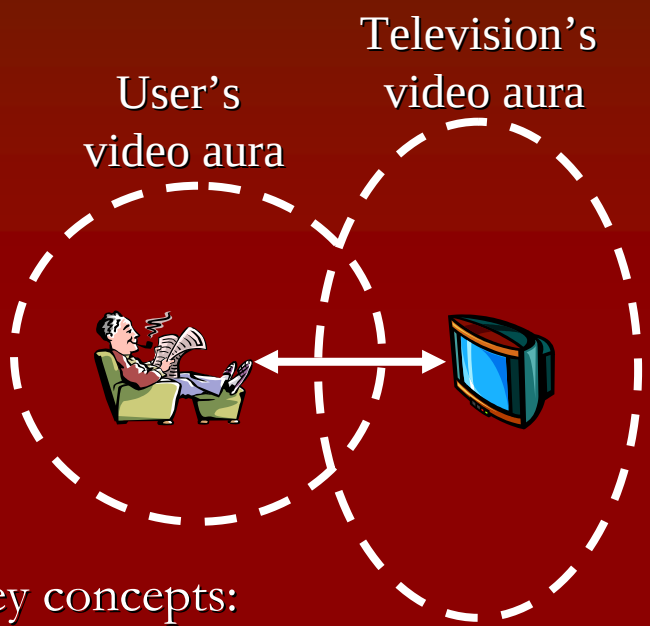
Area-of-interest filtering

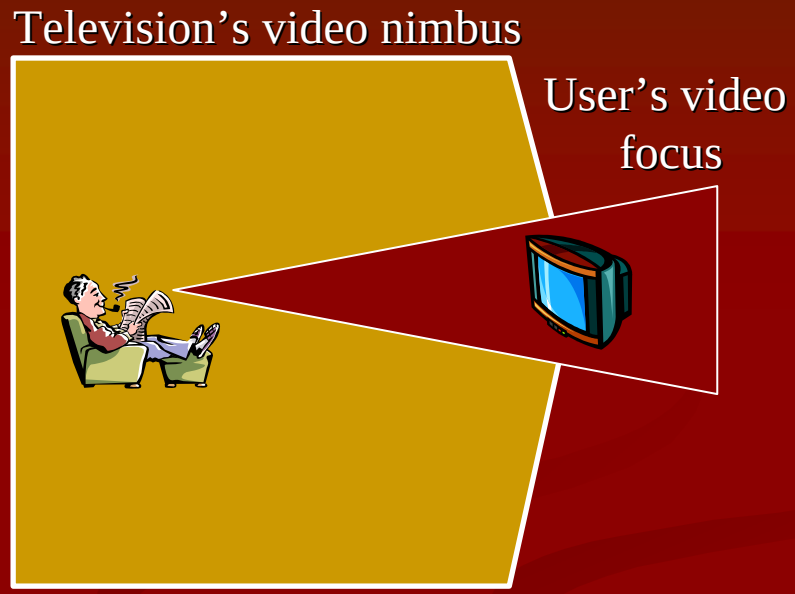
- Area-of-interest filters
  - each host provides explicit data filters
  - filters define the interest in data
- Multicasting
  - use existing routing protocols to restrict the flow of data
  - divide the entities or the region into multicast groups
- Subscription-based aggregation
  - group available data into fine-grained 'channels'
  - hosts subscribe the appropriate channels

Why to do data flow restriction?



Awareness and the spatial model of interaction





Key concepts:

- *medium*: communication type
- *aura*: subspace in which interaction can occur
- *awareness*: quantifies one object's significance to another object (in a particular medium)
- *focus*: represents an observing object's interest
- *nimbus*: represents an observed object's wish to be seen
- *adapters*: can modify an object's auras, foci, and nimbi

Nimbus-focus information model

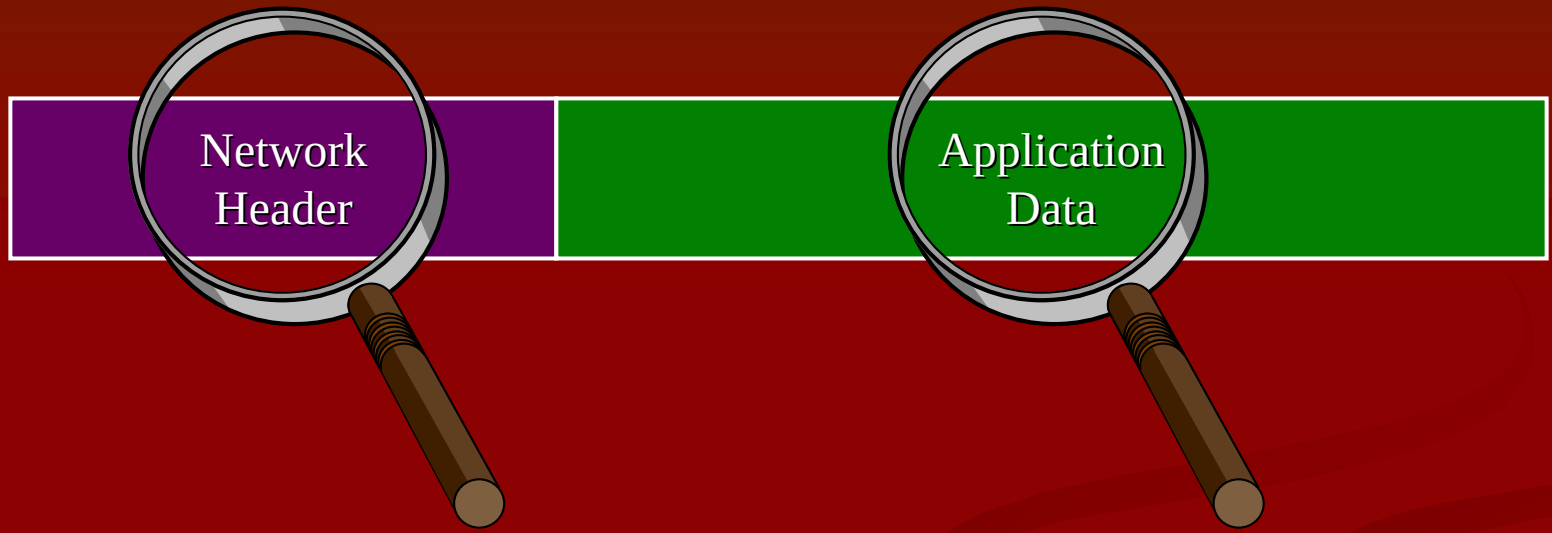
- Nimbus: entity data should only be made available to entities capable of perceiving that information
- Focus: each entity is only interested in information from a subset of entities
- Ideally, all information is processed individually and delivered only to entities observing it
  - what about scaling up?
  - processing resouces
  - each packet has a custom set of destination entities ⇒ hard to utilize multicasting

⇒ Approximate the pure nimbus-focus model

Area-of-interest filtering subscriptions

- Nodes transmit information to a set of subscription managers (or area-of-interest managers, filtering servers)
- Managers receive subscription descriptions from the participating nodes
- For each piece of data, the managers determine which of the subscription requests are satisfied and disseminate the information to the corresponding subscribing nodes
- AOI filtering:
  - restricted form of the pure nimbus-focus model
    - ignores nimbus specifications
  - subscription descriptions specify the entity's focus
  - reduces the processing requirements of the pure model

Intrinsic and extrinsic filtering



Extrinsic filtering

- Filters packets based on network properties
- Implementation efficient
- Filtering cannot be as sophisticated

Intrinsic filtering

- The filter must inspect the application content
- Can dynamically partition data based on fine-grained entity interests

Multicasting

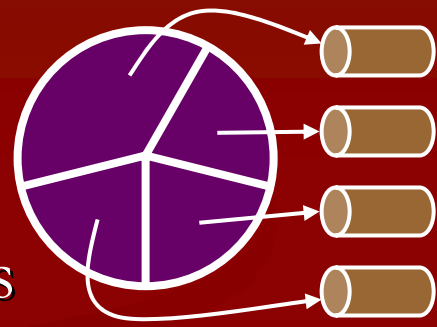
- Transmit a packet to a multicast group (multicast address)
- Packets are delivered to nodes who have subscribed to the multicast group
- Explicit subscription (join group) and unsubscription (leave group)
- A node can subscribe to multiple groups simultaneously
- Transmission to a group does not require subscription
- Challenge: how to partition the available data among a set of multicast groups?
- Each multicast group should deliver a set of related information
- Worst case: each node is interested in a small subset of information from every group ⇒ must subscribe to every multicast address ⇒ broadcast
- Methods:
  - group-per-entity allocation
  - group-per-region allocation

Group-per-entity allocation 1(2)

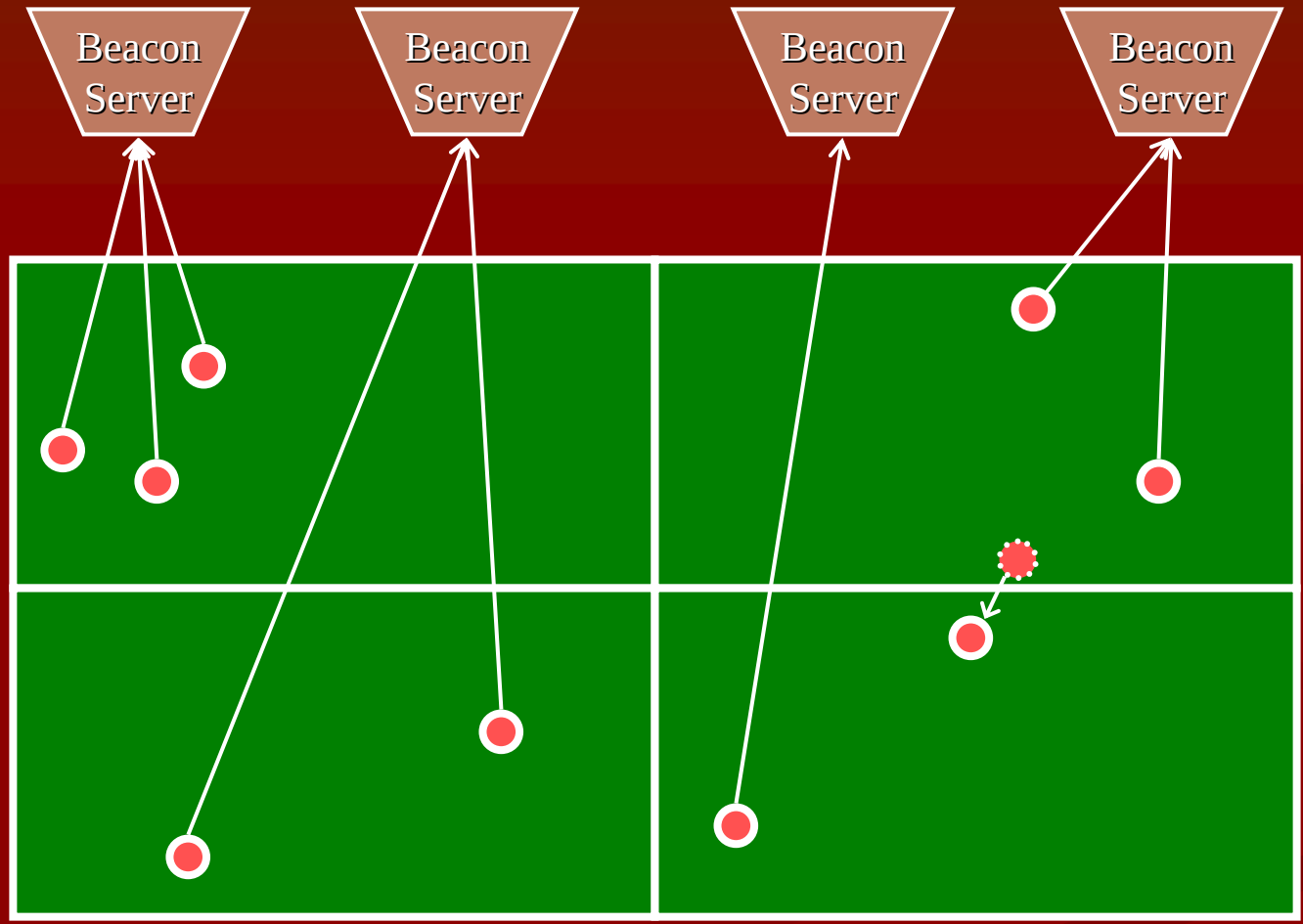
- A different multicast address to each entity
- Each host receives information about all entities within its *focus*
- Subscription filter is executed locally
- Subscribe to the groups which have interesting entities
- Entities cannot specify their *nimbus*; no control over which hosts receive the information
- Example: PARADISE
  - each entity subscribes to nearby entities
  - control directional information interests
    - nearby entities that are behind
    - nearby and distant entities that are in front

Group-per-Entity Allocation 2(2)

- Multiple multicast group addresses to each entity
  - position updates
  - infrared data
- Information at a finer granularity
- More accurate focus by group subscriptions
- Nodes need a way to learn about nearby entities
- *Entity directory service* tracks the current state of the entities
  - entity transmits periodically state information
  - directory servers collect the information and provide it to the entities when requested



Beacon servers



Drawbacks

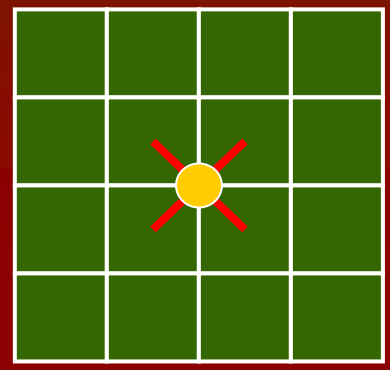
- Consumes a large number of multicast addresses
- Address collisions become quite probable
- Network routers have to process the corresponding large number of join and leave requests
- Group search induces network traffic
- Network cards can only support a limited number of simultaneous subscriptions
  - too many subscriptions ⇒ 'promiscuous' mode

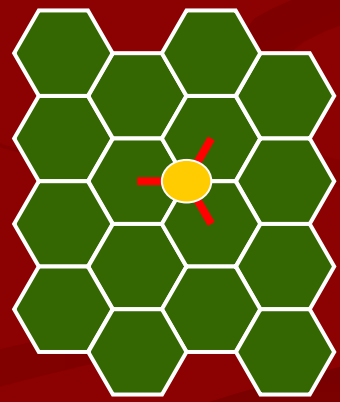
Group-per-region allocation

- Partition the world into regions and assign each region to a multicast group
- An entity transmits to groups corresponding to the region(s) that cover its location
- The entity subscribes to groups corresponding to interesting regions
- Entities have limited control over their nimbus but less control over their focus

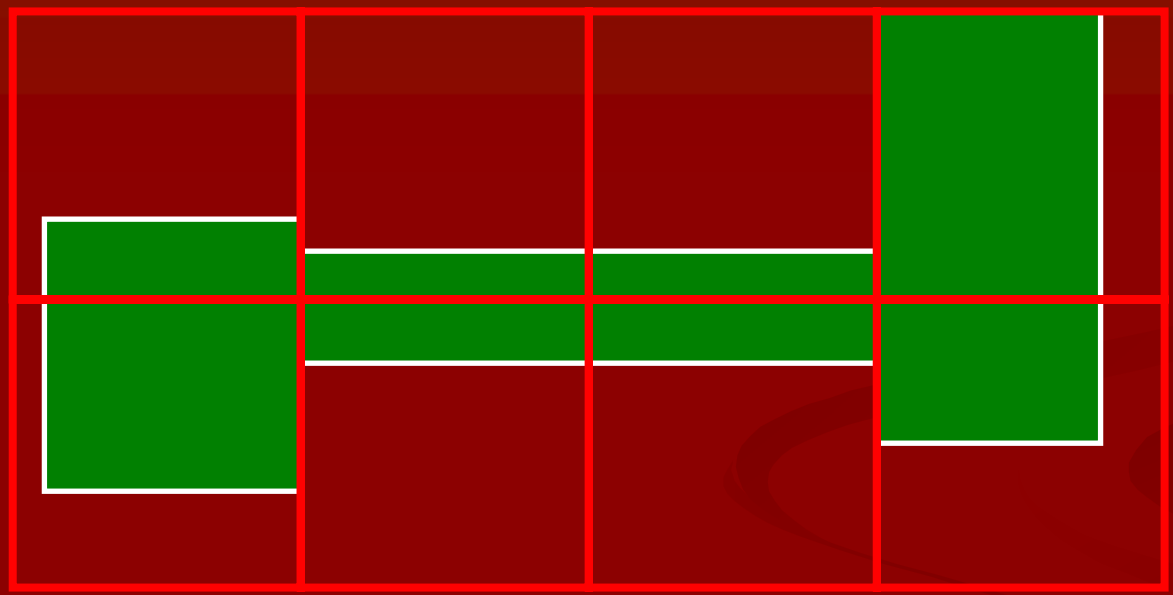
Region bounds

- An entity has to change its target group(s) throughout its lifetime
  - track the bounds of the current region
  - learn the multicast address of a new region
  - boundaries and addresses assigned to the regions are often static
- In grid-based region assignment there are many points at which multiple grids meet
- Near these corners an entity has to subscribe to several groups



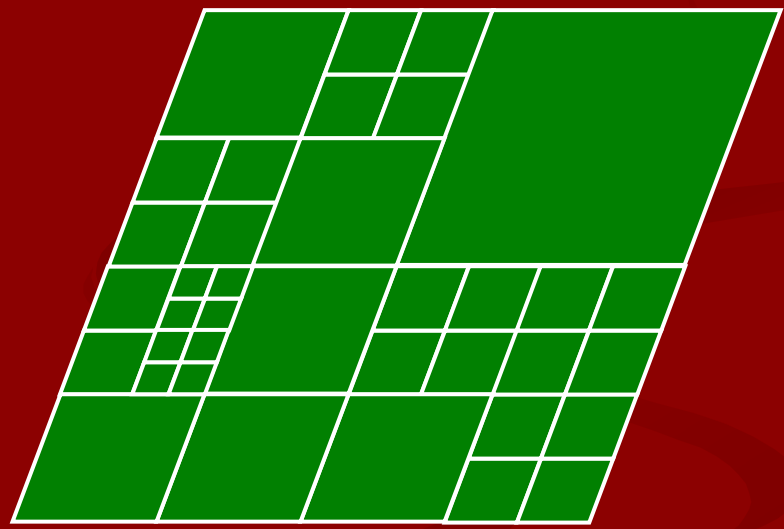


Environment vs. regular tessellation

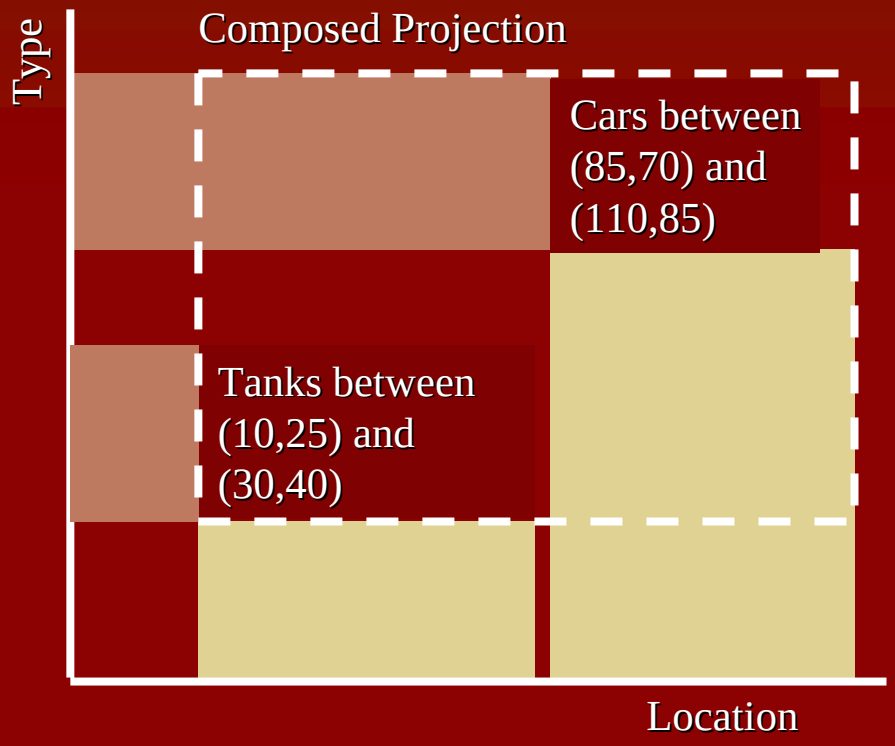


Hybrid multicast aggregation

- Balance between fine-grained data partitioning and multicast grouping
- Three-tiered interest management system:
  1. Group-per-region scheme segments data based on location
  2. Group-per-entity scheme allows receiver to select individual entities
  3. Area-of-interest filter subscriptions



Projections



- Projection aggregation server
  - collect data for a projection
  - transmit aggregated packets (projection aggregations)
- Projection composition
  - merge the interest specifications of the component projections