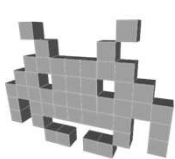
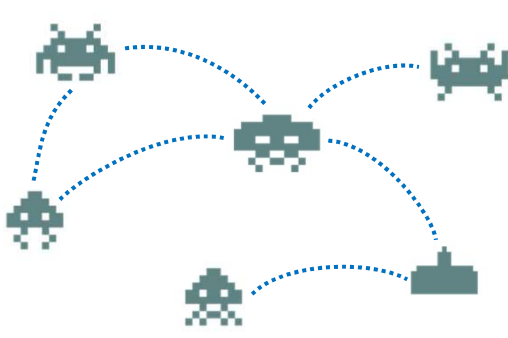


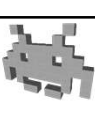
3D VideoGames 2022/2023  
Università degli Studi di Milano

## Networking for 3D Games



1


## Course Plan



- lec. 1: **Introduction** ●
- lec. 2: **Mathematics** for 3D Games ●●●●●●
- lec. 3: **Scene Graph** ●
- lec. 4: Game **3D Physics** ●●●●+●●
- lec. 5: Game **Particle Systems** ◀
- lec. 6: Game **3D Models** ▶●
- lec. 7: Game **Textures** ●●
- lec. 9: Game **Materials** ◀
- lec. 8: Game **3D Animations** ▶●●●
- lec. 10: **Networking** for 3D Games ▶ (highlighted with a red arrow)
- lec. 11: **Artificial Intelligence** for 3D Games ●
- lec. 12: **3D Audio** for 3D Games ●
- lec. 13: **Rendering Techniques** for 3D Games ●

2


## Player 2 has joined the game




- **Multiplayer** game types, according to **gameplay**
  - collaborative
  - competitive
  - versus
  - teams...
- *How much* multiplayer?
  - no: single player
  - 2 players?
  - 10 players?
  - >100?
  - > 1000? } («massively» multiplayer, **MMO**)

3

## Player 2 has joined the game



- Types of multiplayer games
  - Hot-seat
    - players time-share
  - Local multiplayer (Side-to-side)
    - e.g., split screen
    - players share a terminal
  - **Networked**
    - each player on a terminal
    - terminals connected...
      - ...over a LAN
      - ...over the internet



4

## Networking in Games

(see course on: [Online Game Design](#) )

- One task of a Game Engine
- Different scenarios:
  - [number of players](#)? (2, 10, 100, 100.000?)
  - [game pace](#)? (real time action  $\neq$  chess match)
  - [joining ongoing games](#) : allowed?
  - [cheating](#) : must it be prevented?
  - [security](#): is it an issue (e.g. DoS attacks)
  - [medium](#) : LAN only? internet too?  
Latency tolerance? Bandwith tolerance?

5

## Networking in 3D Games

Objective: all players *see* and *interact with*  
a [common](#) 3D virtual world



how can this illusion be achieved?

6

## Dev choices for a networked-game

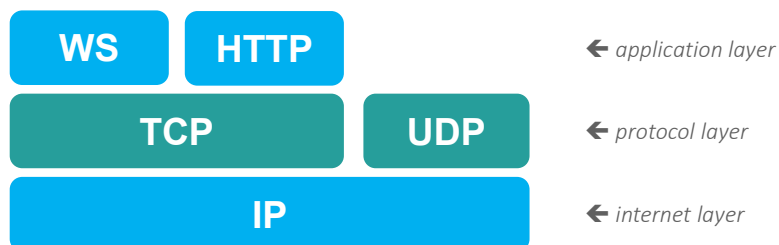


- **What** to communicate?
  - e.g.: complete statuses, status changes, inputs...
- How **often** ?
  - at which rate
- Over which **protocol** ?
  - TCP, UDP, WS ...
- Over which **network architecture** ?
  - Client/Sever, Peer-To-Peer
- How to deal with networking problems
  - **latency** ("lag") <== one main issue
  - limited **bandwidth**
  - connection loss
  - loss of packets

7


## Reminder: Protocols


(see course on: Computer networks )



8


## Protocols



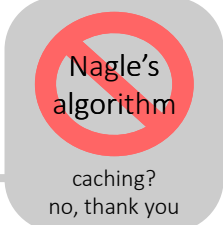
TCP sockets	UDP sockets
<ul style="list-style-type: none"><li>● <b>Connection based</b></li><li>● Guaranteed reliable</li><li>● Guaranteed ordered</li><li>● Automatic breaking of data into packets</li><li>● Flow control</li><li>● Easy to use, feels like read and write data to a file</li></ul>	<ul style="list-style-type: none"><li>● <i>What's a connection?</i> </li><li>● No reliability</li><li>● No ordering</li><li>● Break your data yourself</li><li>● No flow control</li><li>● Hard. Must detect and deal with problems yourself.</li></ul>

9

## UDP vs TCP



- The problem with **TCP**
  - too many strong guarantees
    - they cost in terms of latency (==>lag)!
  - no good for time critical application
    - (if they must be used, at least enable the option `TCP_NODELAY`)
- The problem with **UDP**
  - not enough guarantees
    - guarantee list: "a packet arrives either whole, or not at all". End of list.
    - no concept of connection
      - no timeouts, no handshake, a port receives from anyone
    - no guarantees: packets can arrive...
      - ...out of order :-O, ...not at all :-O, ...in multiple copies :-O



10

## UDP vs TCP



- Problem with **TCP**
  - too many costly guarantees
- Problem with **UDP**
  - not enough guarantees
- The hard way:
  - use **UDP**,  
but *manually re-implement a few guarantees*
  - best, for the most challenging scenario
    - fast paced games, not on LAN



11

## Virtual connections over UDP: how-to (notes)



- add **connection ID** to packets
  - to filter out unrelated ones
- **time out** on prolonged silence (~ few secs)
  - declare "connection" dead
- add **serial number** to packets
  - to detect when one went missing / is out of order / is duplicate
  - (warning: int numbers *do* loop – solutions?)
- give **ack** back for received packets
  - optimize for lucky (& common) cases!
    - $N$  (say 100) received msg == 1 ack (with bitmask)
  - resend? only a few times, **then give up (data expired)**
- congestion avoidance: measure **delivery time**
  - tune send-rate (packets-per-sec) accordingly
- obviously: **NON blocking** receives!

what **TPC**  
doesn't  
understand



12

## Choosing a Protocol

- In summary, it is a question of pacing
  - fast paced game?
    - action games, FPS, ...
    - (sync every 20-100 msec)

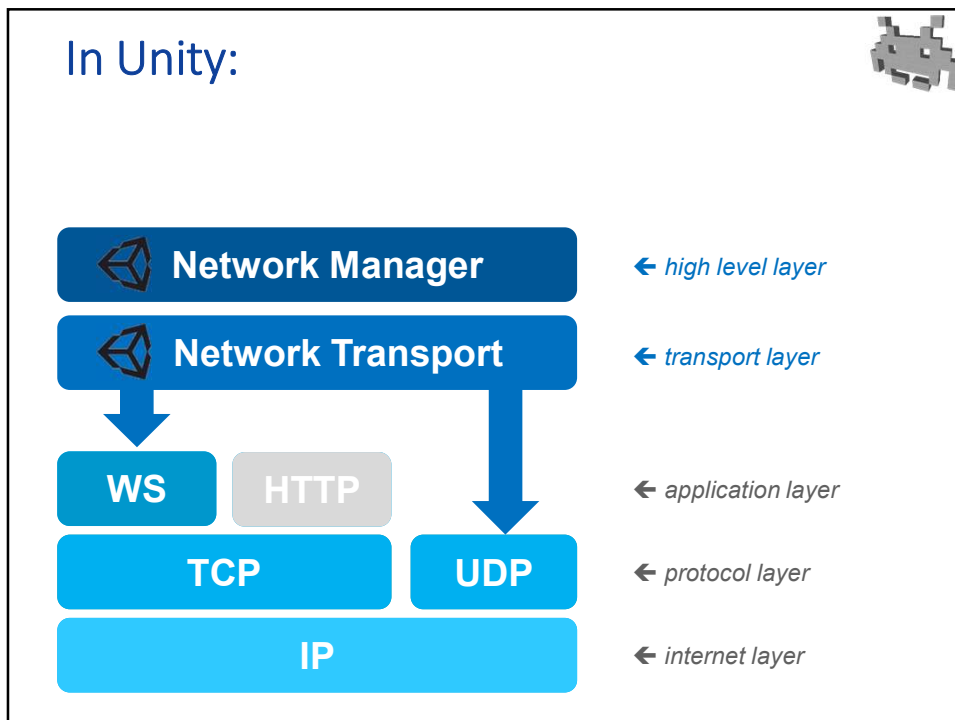
→ UDP necessary (unless LAN only)
  - slow paced game?
    - RTS, RPG...
    - (sync every ~500 msec)

→ can get away with TCP
  - slower paced games?
    - MMORPGs, cards ...
    - (sync every few sec)

→ why not just HTTP
  - traditional turn based ?
    - chess, checker
    - (sync every hour/day)

→ may as well use EMAIL

13



14

## In Unity:

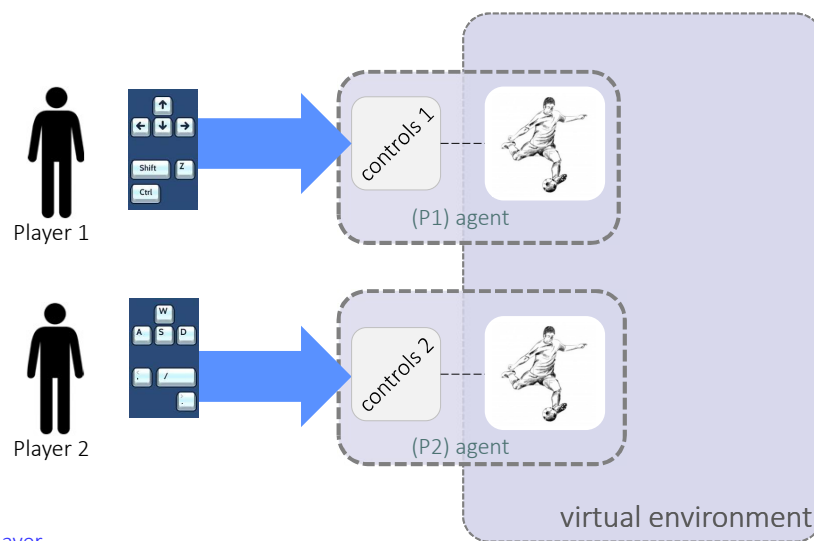
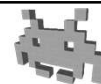


- Low level: **Transport Layer**
  - Builds up guarantees over UDP (connections)
  - Easy to use as TCP, but optimized for games
    - see how-to list above
  - Can work over WS instead UDP (abstracts the differences)
    - WS needs be used for web / WebGL games
- Hi level: **Network Manager**
  - presets network connectivity
  - standard “client hosted” games
    - server is also a player
  - controls shared state of the game
  - deals with clients
  - sends remote commands

15

## Controls and Agent

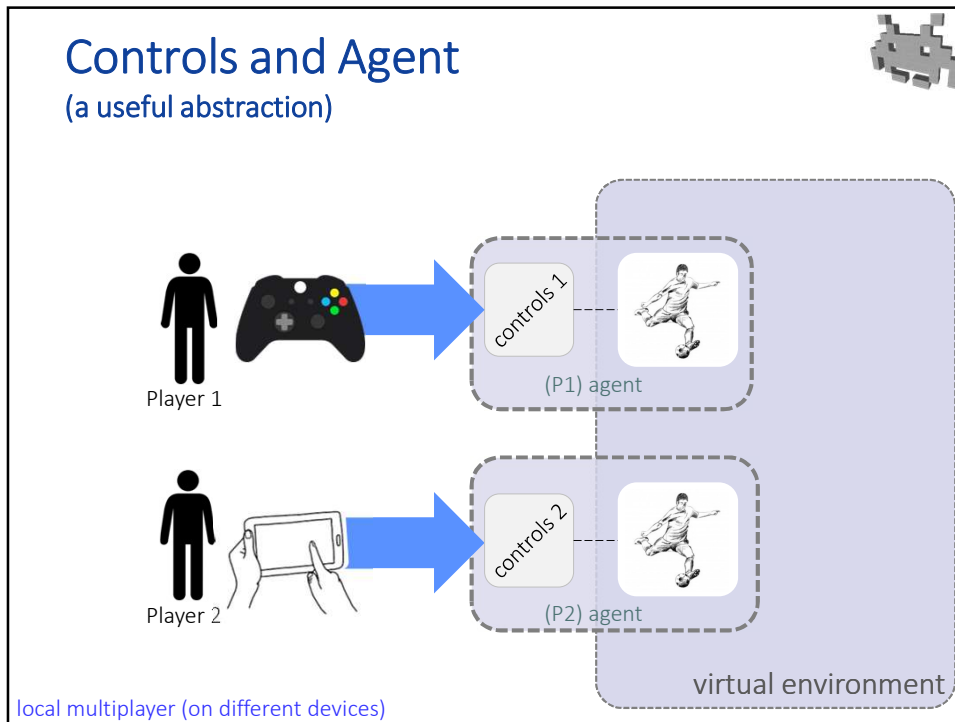
(a useful abstraction)



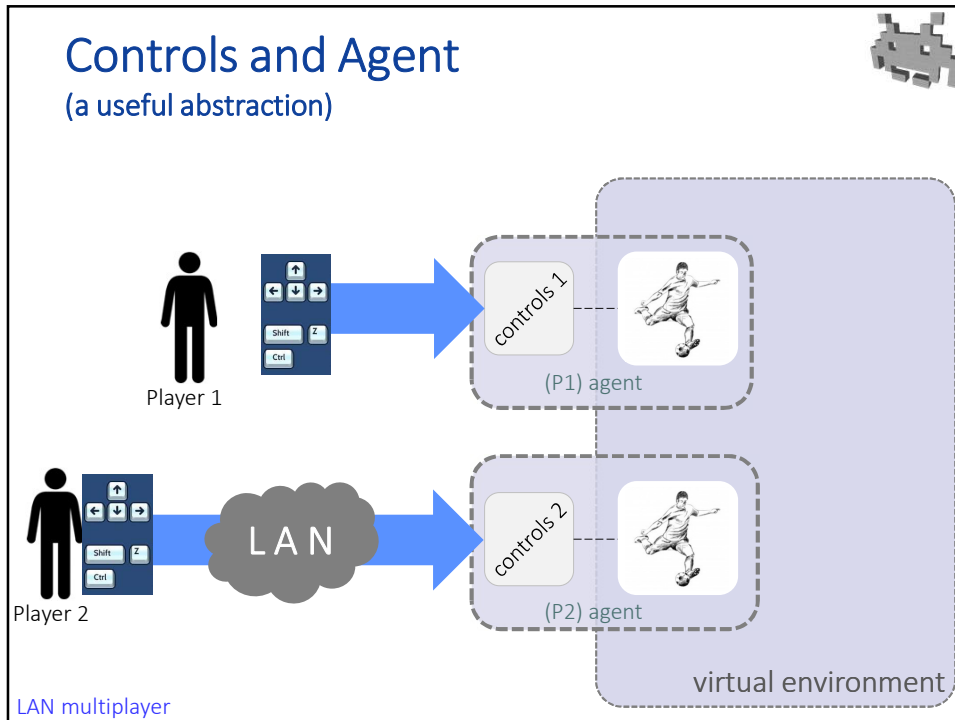
local multiplayer

16

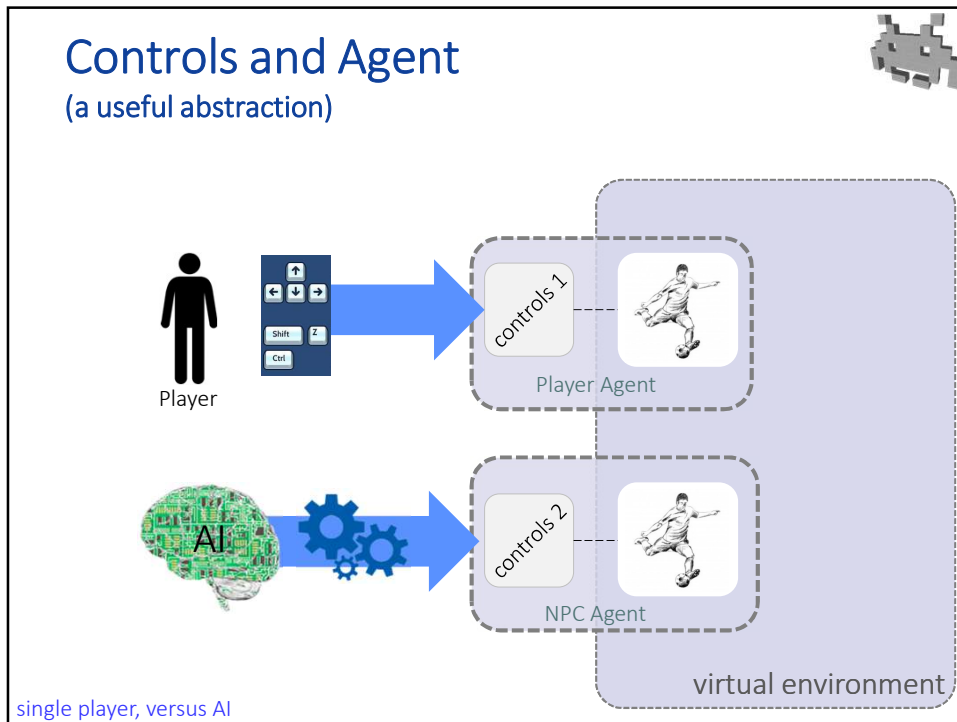




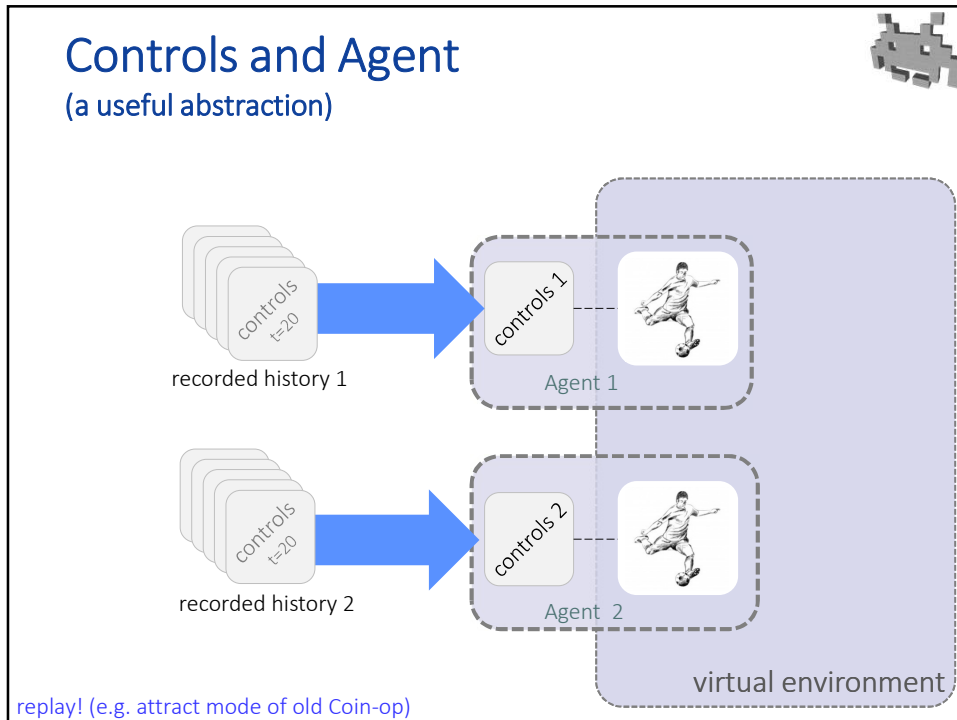
17



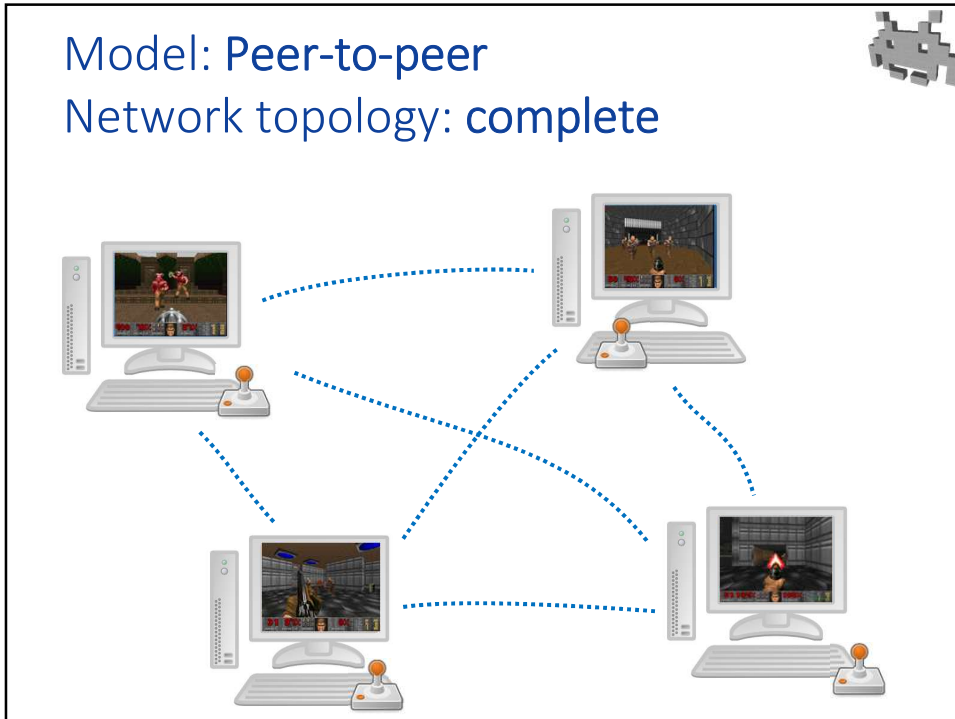
18



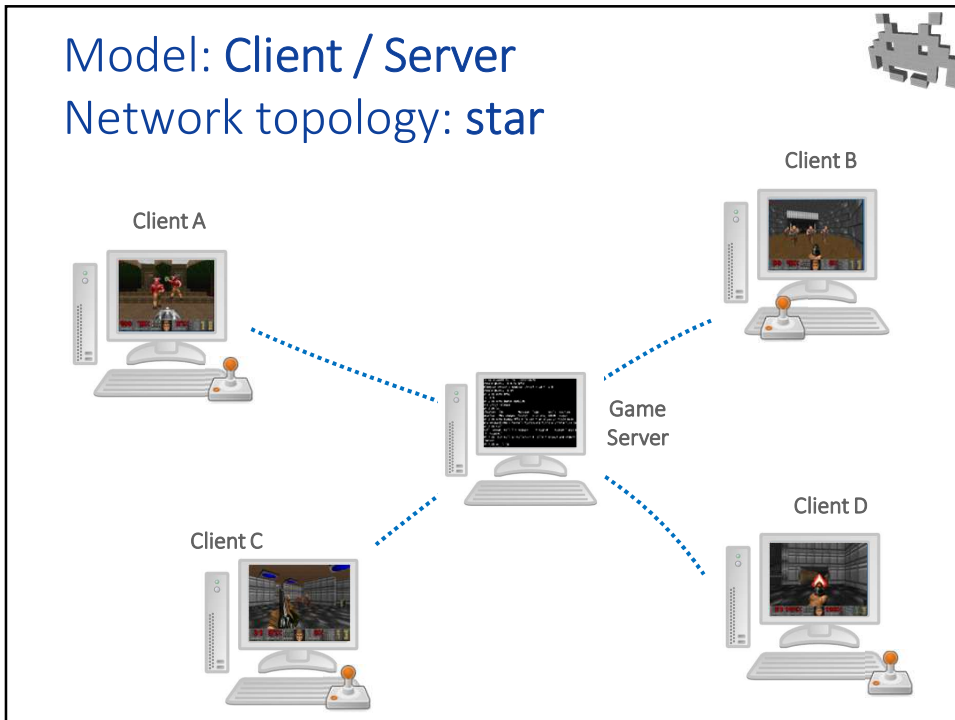
19



20



21



22

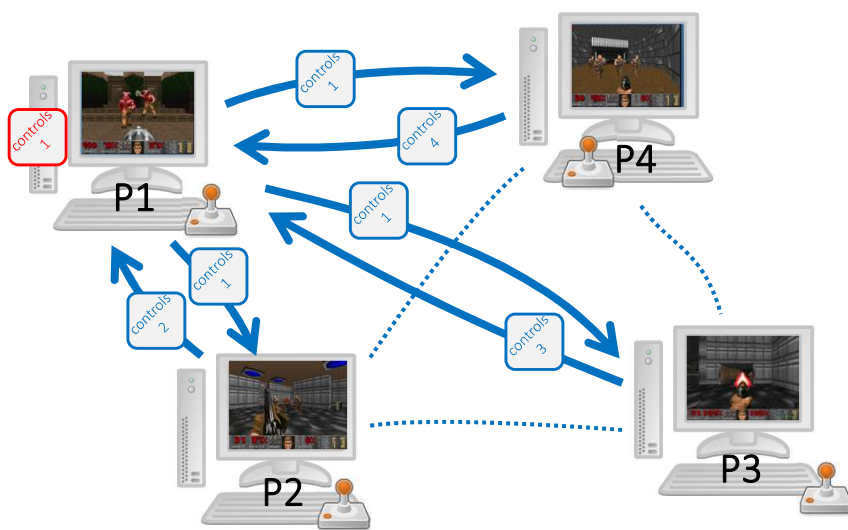
## Networking paradigms for games we will discuss

- Deterministic Lockstep on P2P
- Deterministic Lockstep
- Game-Status Snapshots
- Distributed Physics (just notes)
- Game-Status Snapshots with Client-Side predictions
- Cloud gaming

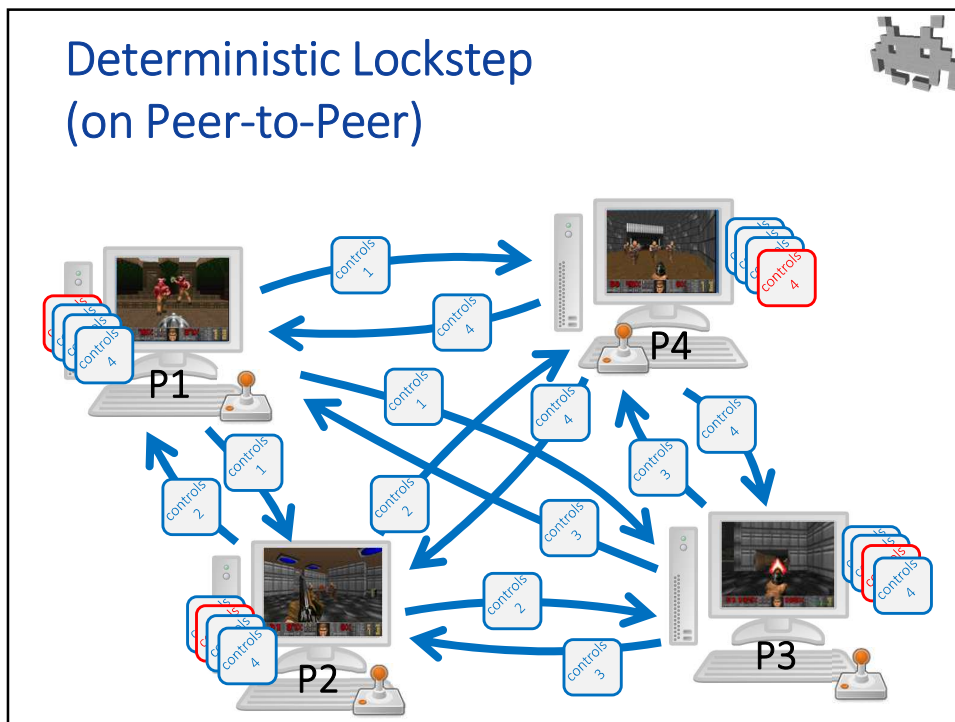
on client-server

23

## Deterministic Lockstep (on Peer-to-Peer)



24



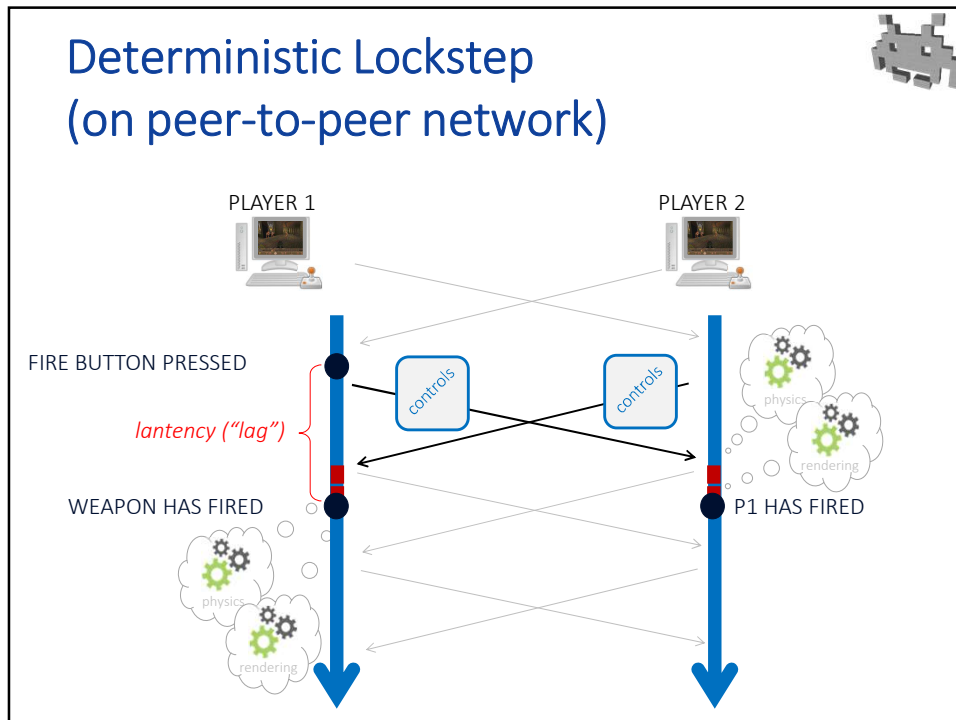
25

### Deterministic Lockstep (on Peer-to-Peer)

- Game evolution = sequence of “turns”
  - e.g. physics steps (fixed  $dt$  !)
- Each node sends its current *controls* (inputs)
  - to everybody else
- After all controls are received, each node computes its own evolution
  - deterministically:  
same input → same result

← even if independently computed

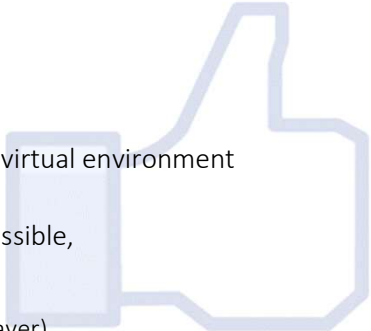
26



27

### Deterministic Lockstep: the good

- elegant and simple! 😊
- minimal **bandwidth** needed
  - only sent data = controls
    - compact! (e.g., a bitmask)
  - does not depend on complexity of virtual environment
- **cheating**: inherently limited
  - but a few ways to cheat are still possible, e.g.:
    - aim-bots (unlawful assist from AI)
    - x-rays (unlawful reveal of info to player)
- mixes well with:
  - non-cheating AI, replays, player performance recording...
- can use simple **TCP connections**
  - because we need 0% packet loss anyway (but...)



28

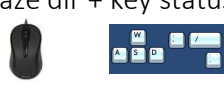
## Deterministic Lockstep: can as well use TPC instead of UDP ?

- why yes:
  - TPC is simple to use
    - It takes care of everything
  - works well, when no packet loss
    - on loss, we need resend it anyway: let TPC do that
  - makes little sense to use UDP and then... try to re-implement all TPC over it
  - at the beginning of dev, UDP is a (premature) optimization
- why not:
  - to degrade better with packet loss
  - e.g.: use redundancy – instead of resend-on-failure
    - controls are small: send 100+ controls in every packet
    - keep resending until ack received


29

## Deterministic Lockstep


- Common, e.g., in:
  - RTS
    - controls = orders
      - can be fairly complex
      - but game status = much more complex
  - first generation FPS
    - controls = [ gaze dir + key status ]



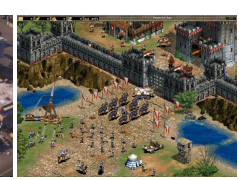
...why not anymore?




Starcraft Blizzard 1998-2015



Command and Conquer EA / Westmany et al 1995..2012





Age of Empires Ensemble Studios et al, 1998..2015



Doom ID-soft, 1998

31


## Deterministic Lockstep on peer-to-peer: the limitations



- **responsiveness:**
  - input-to-response delay of 1 x delivery time (even locally!)
  - (you cannot act immediately even on your own local input)
- **does not scale** with number of players
  - quadratic number of packets
  - 2P ok, 100P not ok
- **input rate = packet delivery rate**
- **delivery rate** = as fast as the *slowest* connection allows
- if connection problems (anywhere): everybody freezes!
- joining ongoing games: difficult to implement
  - needs sends full game state to new player
- assumes full agreement on initial conditions
  - this is not problematic
- **assumes complete determinism!**
  - this can be problematic (see next slide)

32

## Determinism: what can break it



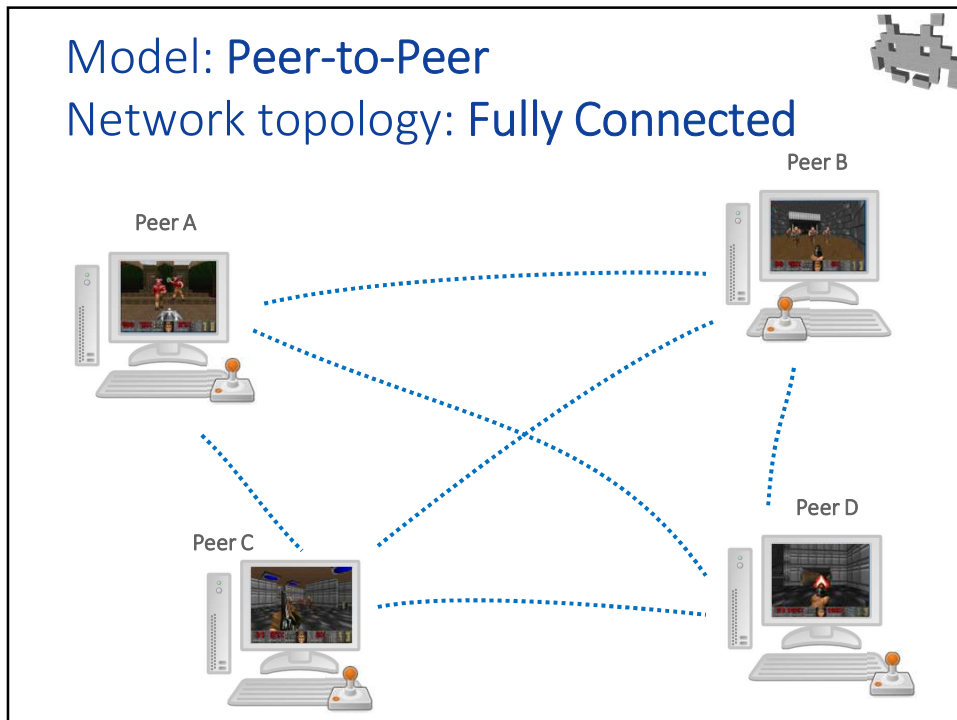
- Pseudo-**Random**? → not as problem
  - fully deterministic (just agree on the seed)
- ⚠ **Physics:** many preclusions and traps
  - ⚠ variable time step? **bad**
  - ⚠ time budgeting? **bad**
  - ⚠ hidden threats:  
order of processing of particles/constraints
- ⚠ anything that depends on **clock**?  
→ **poison** to determinism
- ⚠ **GPU** computations? **very dangerous**
  - slightly different outcome on each GPU model
- ⚠ **floating point** operations?
  - **hidden dangers**,
  - different hardwired implementations
  - best to assume very little (**fixed point** is 100% safe)
- ⚠ **NOTE:** 99.999% correct == **not correct**
  - virtual world is faithful to reality enough to be *chaotic* → butterfly effect:  
the tiniest local difference == expect completely different outcomes soon

The entire game system must be designed from the start with “determinism” in mind ...

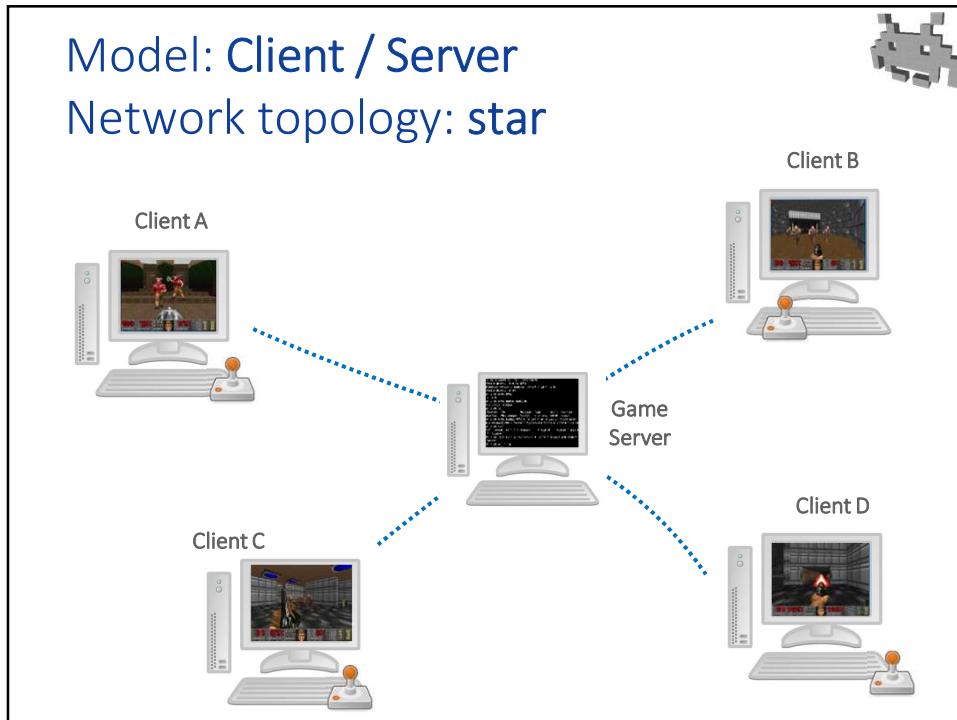
...and still, it difficult to get (and debug)

33





34



35

## Deterministic lockstep on Client-Server



- Server sits on the central node
- Protocol:
  - Each client sends his controls to server
  - Server collects all controls and sends them back to clients
- Advantage:
  - scalability:  
number of packets is linear (not quadratic)
- Cost:
  - **responsiveness:**  
**latency = 2 × delivery time :-O**
- Bonus: the server can now be made **authoritative**
  - Many new options available. For example...

hurts  
gameplay!

36

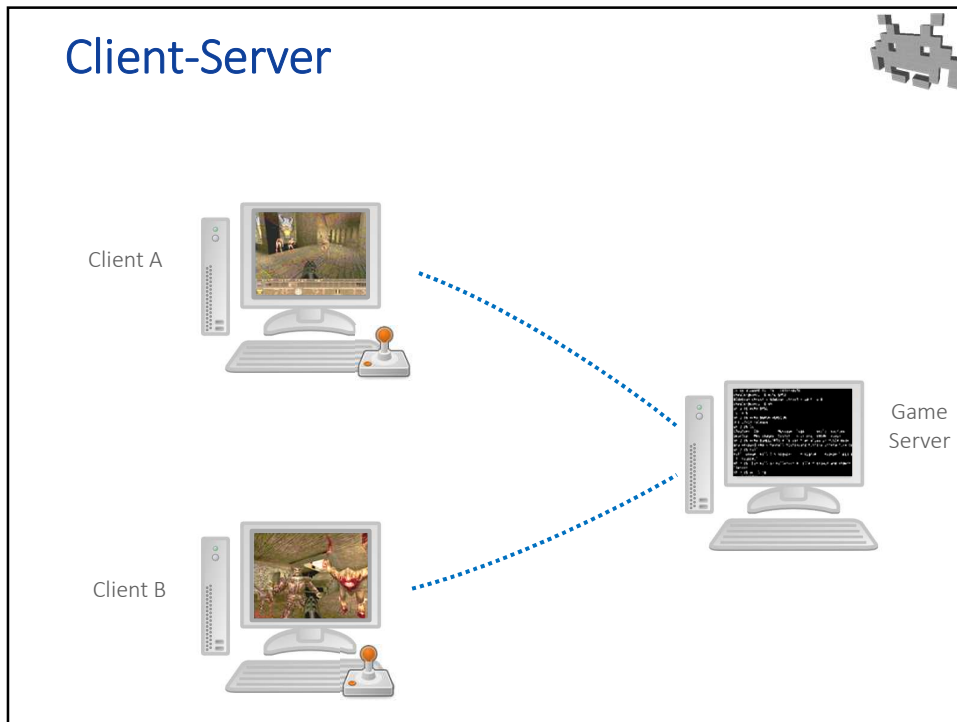
## “Server is the man” \* (authoritative server)



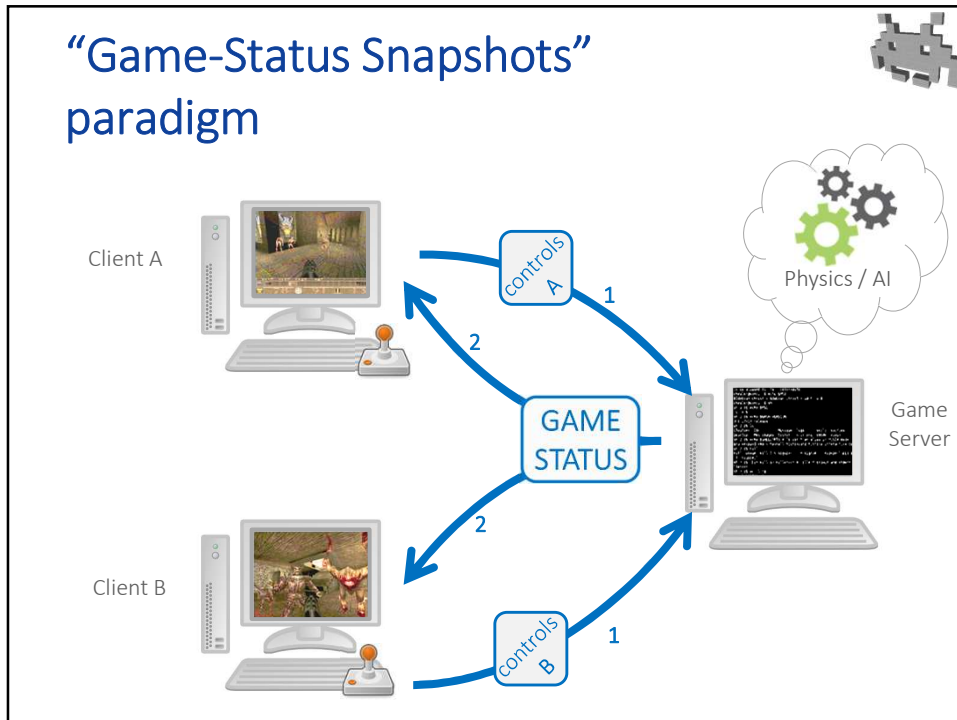
- The server has the last word
- For example:
  - Packet loss from player 3?  
Server makes up controls for player 3  
(instead of waiting for them)
  - Note: server *defines* what player 3 did,  
not player 3 itself!
  - i.e., clients take server’s word even for their own actions
  - Packet loss affects one player only

\* Tim Sweeney (Unreal)

37





38



39

## Game-Status Snapshots






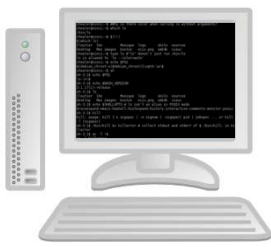
**Client:**

- just a remote visualizer of the current status
  - status is “read only”
- remote input collector

**Server:**


- computes evolving status
  - including physics
- it’s where the “real game” runs





41

## Game-Status Snapshots



● **Client:**

- connected: to server only
- captures input
- sends controls
- receives game status
  - or relevant portions of it
- renders it
  - using all relevant assets

● **Server**

- connected: to all players
- receives all controls
  - missing? doesn’t matter
- updates game status
  - physical simulations, etc
- sends current status
  - to all

Physics, cosmetic effects only

Graphics

Sounds   UI

Physics

AI

Scripts

42

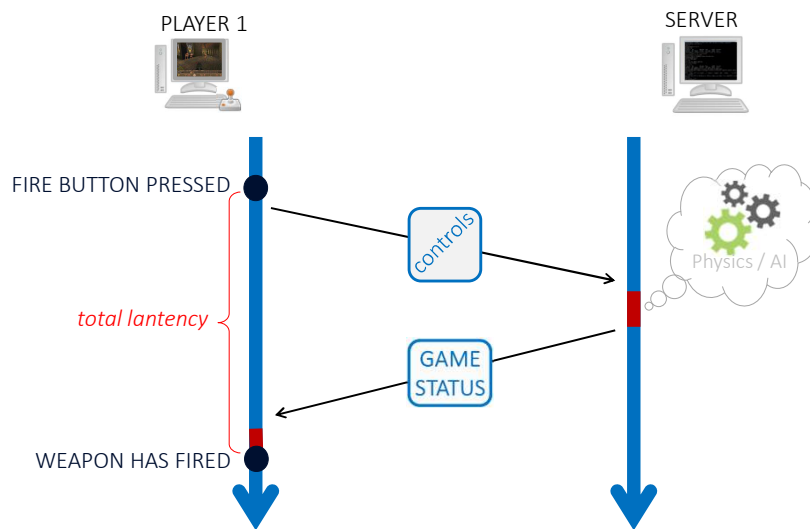
## Game-Status Snapshots (compared to deterministic lockstep)

- the gains:
  - determinism: not a required assumption anymore
  - joining ongoing games: becomes trivial
  - packet loss: bearable (hurts that player *only*)
    - to profit: use **UDP**
  - slower connection: bearable (affects that player *only*)
- the losses:
  - packet size: a lot bigger! optimizations, to counter this:
    - compress world status
    - send only the portions of the status which changed or which interest a player
  - **perceived latency:**
    - from input to effect = delivery time :-(
    - from input to visual = 2 x delivery time :-O

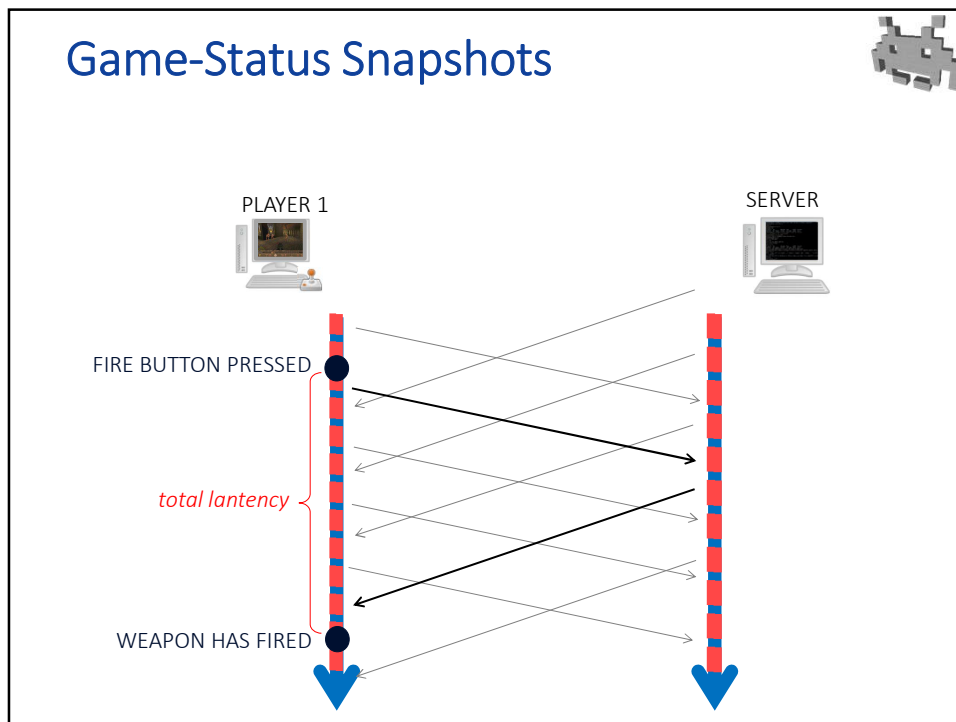
Loss of responsiveness hurts gameplay!

43

## Game-Status Snapshots



44



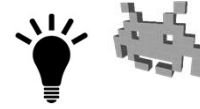
45

### Game-Status Snapshots: with Interpolation: the idea

- World “Snapshot” contains:
  - The scenegraph (local transforms per object)
  - Anything else needed e.g. for gameplay
- Problems:
  - snapshot size can be large! (even with optimizations)
  - few FPS (in the physical simulation)
  - ==> “jerky” animations
- Solution 1: client-side **interpolation**
  - client keeps last two snapshots in memory
    - last received one + the previous one
  - interpolates between them,
    - client lags behind server by even more!
  - gain: smoothness (high FPS with low packet - rate)
  - loss: responsiveness (increased latency) **oh noes!**

46

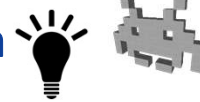
## Game-Status Snapshots: with Extrapolation: the idea



- World “Snapshot” contains:
  - data needed for 3D rendering:  
(position-orientation of objects, plus anything else needed)
- Problems:
  - large snapshot size! (even with optimizations)
  - few FPS (in the physical simulation)
  - ==> “jerky” animations
- Solution 2: client-side **extrapolation**
  - clients keeps last two snapshots in memory
    - last received one + the previous one
  - extrapolates between them, i.e., shows the expected “future”
    - i.e. it shows an attempted prediction to the next snapshot
    - NOTE: this prediction is often wrong: glitches.
  - gain: responsiveness
  - loss: accuracy - lots of glitches. :-{

47

## Partial Client-side Game Evolution (aka *distributed physics*): the idea

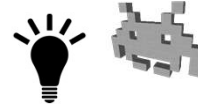


- Each client:
  - in charge for game evolution
    - including physics
  - communicates to others a reduced game-status snapshot
    - describes only status of own player  
(e.g. it's transforms, transforms for its flying bullets, etc)
  - receives other partial snapshots
  - merges everything up
    - (updates statuses of other players)
- Simple, zeroed latency
  - immediately responsive to local player controls
  - remote agents updated according to “what their client says”
- Problem: can still need determinism
  - (who keeps NPCs / environment in sync?)
- Problem: **authoritative clients**: prone to **cheating**!!!

to server,  
or , in a P2P network,  
to each other peers

48

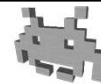
## Client-Side Prediction: the idea



- Client:
  - get Commands from local inputs
  - sends Commands to Server
  - computes game evolution (the prediction)
    - maybe “guessing” other players commands (which it ignores)
    - zero latency!
- Server:
  - receives Commands (from all clients)
  - computes game evolution (the “reality”)
    - server is authoritative
    - prevents many forms of cheating
  - sends Snapshot back (to all clients)
- Client:
  - receives Snapshot (the “real” game status)
  - corrects its prediction, *only if needed*

49

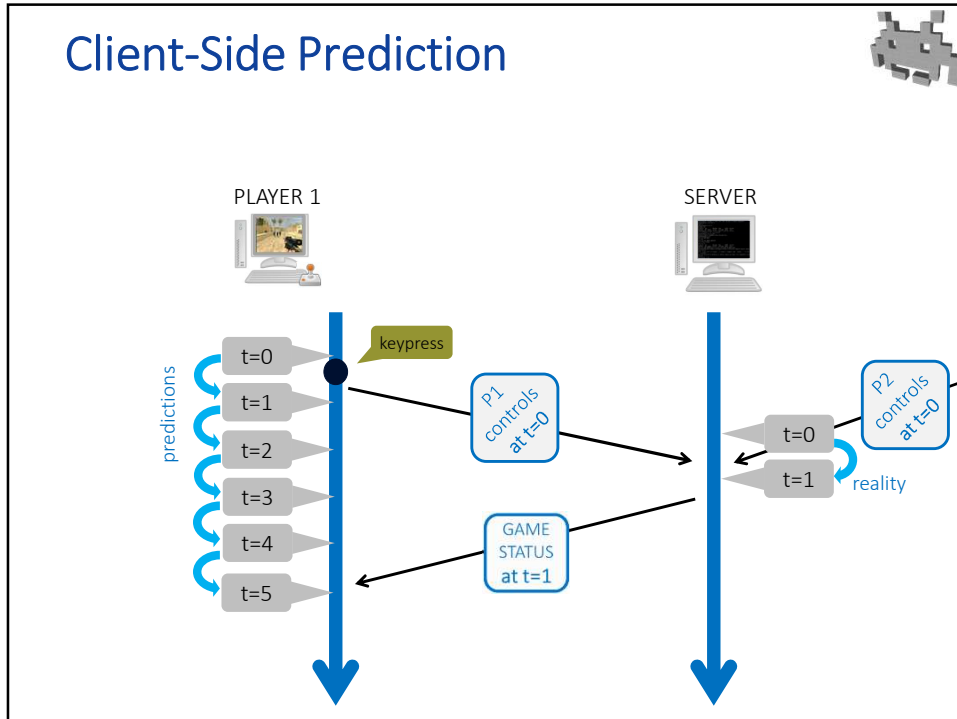
## Client-Side Prediction with corrections from the server



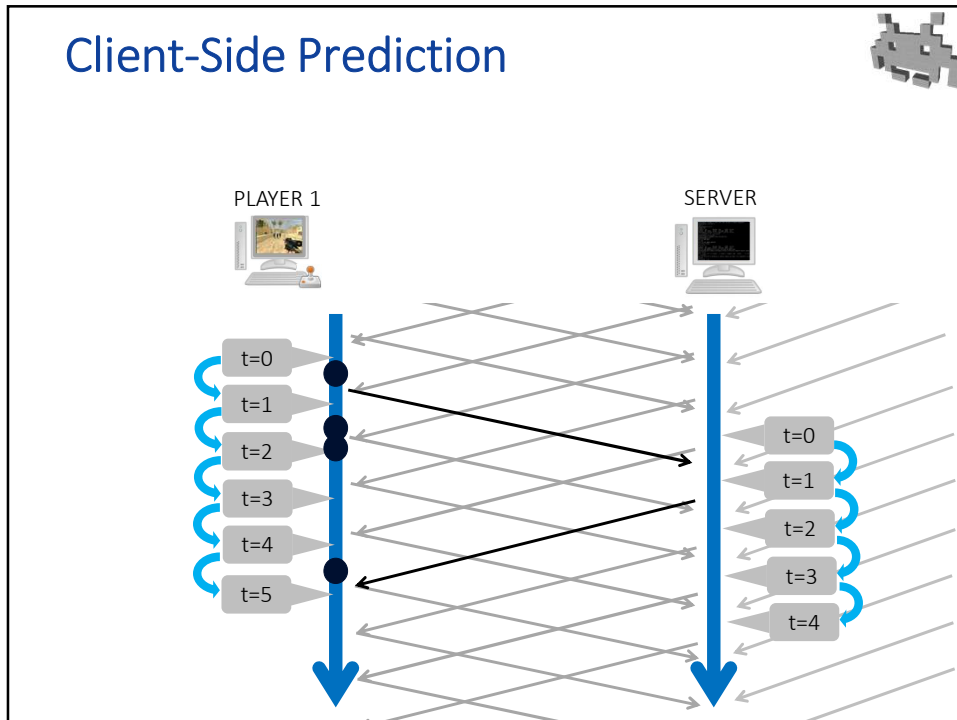
- The server-side “real” simulation lives  $k$  msec in the past of the client-side “predicted” one
  - $k$  = deliver time
  - remember: virtual time != real world time
- When server correction arrives to client, it refers to  $2k$  msec ago (for the client)
- Q: how to correct... *the past?*

50





51



52

## Client-Side Prediction: correction from the server



- Q: How to correct... *the past*?
- A:
  - keep last  $N$  statuses in memory
    - including own controls
  - as the “real” status (the correction) of the past arrives from server...
  - ...compare it with stored past status (at corresponding time):
    - does it **match**? ← maybe: within some tolerance  
nothing to do
    - does it **mismatch**?  
discard frame *and following ones*,  
rerun simulation to present (reusing stored controls)

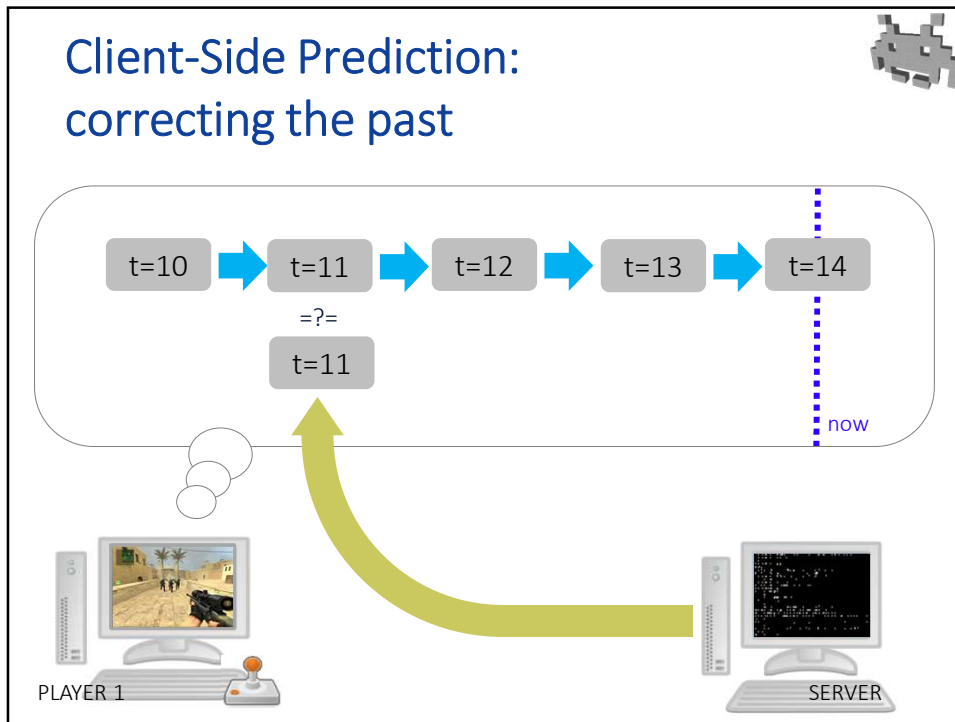
53

## Re-running physical simulation

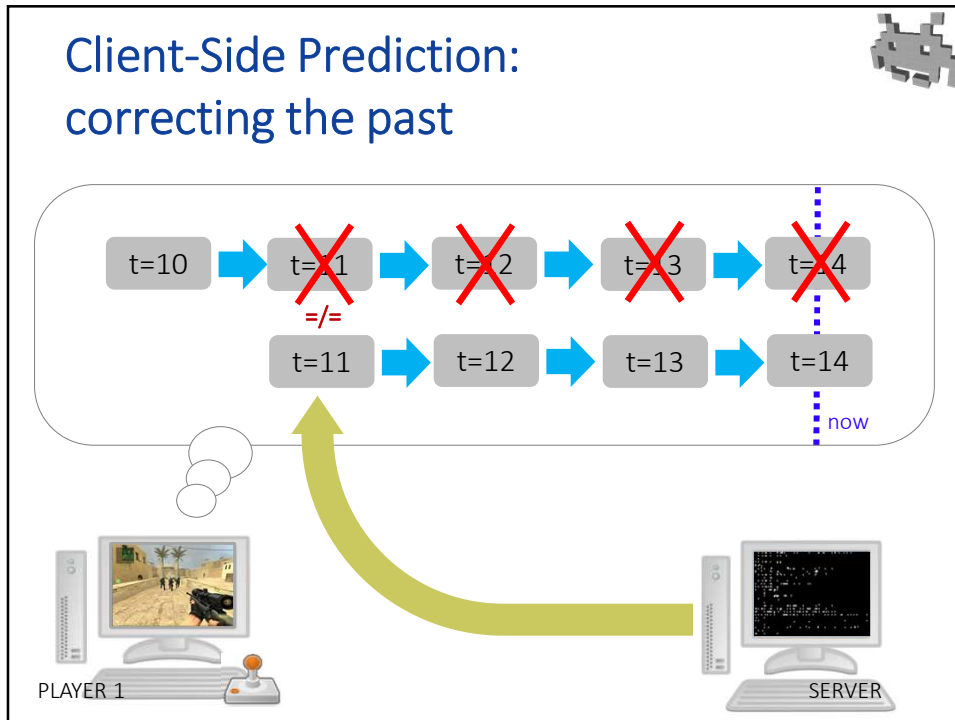


- We just need to catch up with the present
- Physics and AI only
  - no graphics, no sound rendering,  
no cosmetic physics, particle system...
- At full speed: can use larger  $dt$  if necessary
  - This only compromises accuracy a bit -- tolerable
- Must reuse same controls of own player and other's
  - They must all be cached also
- Note: player is never shown these intermediate steps; only the final result
- The price to be paid: glitches when switching from current present to a different (corrected) present

54




55



56

## Client-Side Prediction: what causes mispredictions?




- Lack of **determinism**
  - e.g., physics was approximated – *soft* real-time!
  - see above for more possible causes of this
- Didn't account that *own* controls were **not received** by server (in time)
  - server: "actually, back then, you didn't jump"
  - authoritative server – server *defines* the truth (even when the client is in a better position to know)
- Didn't account for **other players' controls**
  - server: "actually, back then, that other player jumped"
- Note: none of the above breaks gameplay too much
  - it just causes minor / temporary glitches (usually)

minor/rare issues

commonest cause!

57

## Client-Side Prediction: optimizations 1/2



- reduce snapshots size  
(==> to increase packet frequency)
  - partial snapshots: refresh more often the parts which are most likely to be predicted wrong / or which changed
  - drastic space reductions!
  - but make sure that every part is eventually refreshed
- reduce correction computation  
(==> making corrections quicker)
  - partial physic steps:  
update only the parts affected by the error
  - use bigger dt (fewer steps to get to present)

58

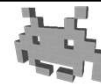
## Client-Side Prediction: optimizations 2/2



- tentatively predict also unknown data  
(=> to reduce correction frequency)
  - e.g., also predict other player's controls
  - easiest prediction: players do what they did last frame
- trigger correction only when status differ *enough*  
(=> to reduce correction frequency)
  - e.g., when any spatial position difference > epsilon
  - tolerate small discrepancies
  - (warning: discrepancies tend to explode exponentially with virtual time – because Chaos)

59


## Client-Side Prediction: notes



- A snapshot = includes physical data
  - not just for the 3D rendering, also to update physics
  - can be made smaller, with optimizations
- 😊 No **latency**: immediately react to local input
  - client proceeds right away with next frame
  - *when prediction is correct*: seamless illusion
  - *otherwise*: (minor?) glitches
- 😊 **Determinism**: not assumed
- 😊 **Cheating**: not easy (server is **authoritative**)

60


## Summary : rules of thumb



- Comparing network layouts
  - **peer-to-peer** :
    - ☺ reduced latency
    - ☹ quadratic number of packages (with number of players)
  - **client-server** :
    - ☹ doubled latency
    - ☺ linear number of packages (with number of players)
    - *REQUIRED*, for any solution with a **authoritative server**
    - *REQUIRED*, for num players >> 4-6

61

## Summary : rules of thumb



- Comparing network paradigms
  - **Deterministic Lockstep**, if
    - determinism can be assumed
    - few players (up to 4-5)
    - fast + reliable connection (e.g., LAN)

← **RTS** most common option !

} *or, slow paced game*
  - **Game-status Snapshots**, if
    - game status not overly complex
    - a little latency can be tolerated
  - **Client-side evolution**, if
    - preventing cheating not important
  - **Client-side prediction + server correction**, if
    - game status not overly complex

↓ **FPS** most common option !

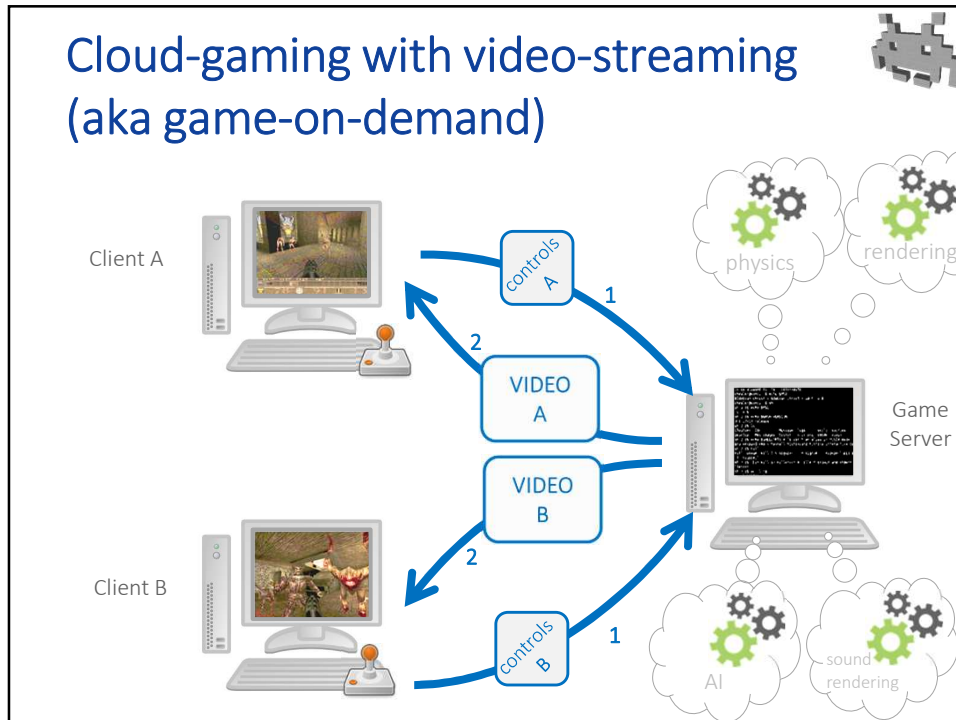
62

## Summary: classes of solutions

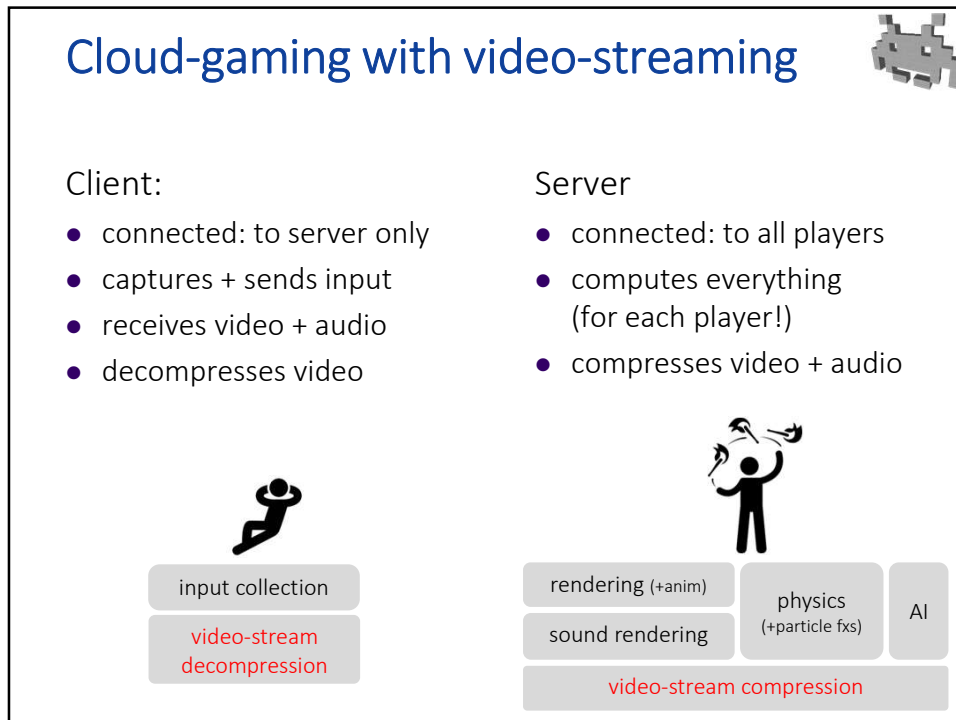
- Who computes game evolution? (including physics)
  - **deterministic-lockstep** : clients
    - there may be no server at all: peer-to-peer
    - independent computations, same result
  - **game-status snapshots** : server
    - clients are just visualizers
    - maybe with interpolation / extrapolation
  - **(distributed physics** : both clients and server)
    - clients in charge for own agent(s)
    - server in charge for env. / NPCs
  - **client-side predictions** : both clients and server
    - clients “predict” (just for local visualization purposes)
    - server “corrects” (it has the last word!)

63

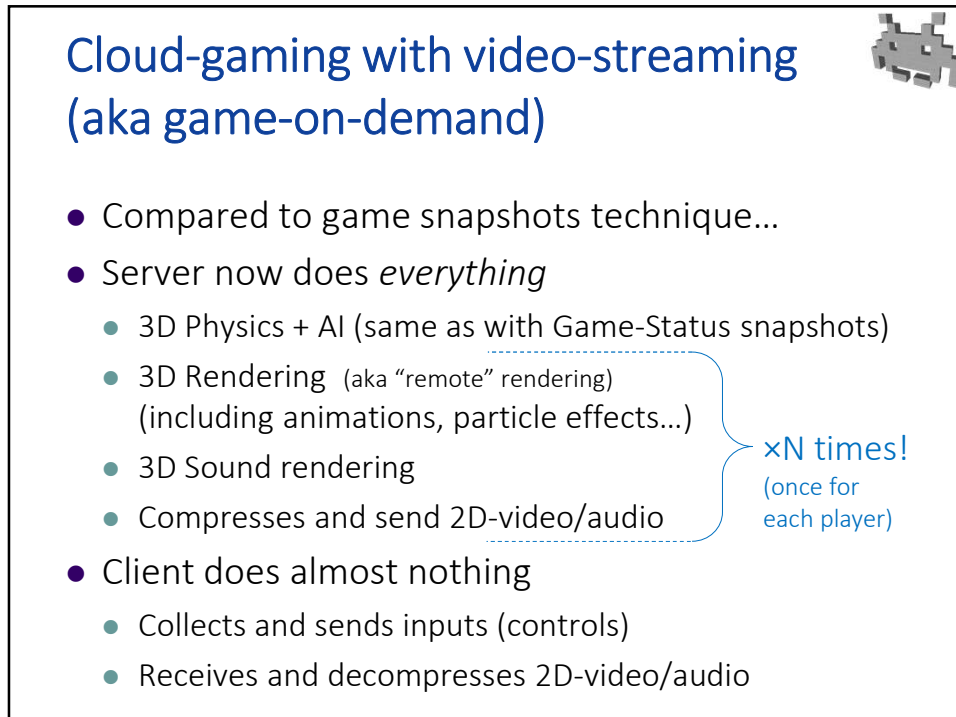
## Cloud-gaming with video-streaming (aka game-on-demand)



64



65



66



## Cloud-gaming with video-streaming (aka game-on-demand)



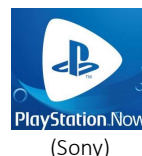
- Advantages: client is thin
  - client does almost nothing
  - client needs nothing (no asset, no storage, no DRM)
  - needed capabilities are limited (pads, cellphones ok)
- Challenges:
  - Demanding in terms of bandwidth (high-res video + audio)
  - Demanding in terms of server workload
  - **Latency!** Impossible to reduce or hide (by prediction), plus compression by server, plus decompression by client
    - luckily, video-on-demand technologies can be leveraged
  - Video resolution, FPS: are very costly

67

## Cloud-gaming (aka gaming-on-demand)



- A heavily invested-on, fast-growing approach to 3D game networking
- Latency = maybe 80-120 ms
  - Is this acceptable?
- Bandwidth = maybe 5-50 mbits/s
- Will it become an established platform for 3D games?



68