Ping Pong Game

Sprint 2

Group 326
Yuanyuan Dong (ydong1)
Tianyi Chen (tianyic1)

Catalog

Part I

Part I Background

Background

Overview of the project

Our project is a web-based online 2-D ping pong game platform which can support multiple users.

In Sprint 1

- Building the overall website architecture including user registration, login, verification
- Building game coordinating logic, on top of Django channel group

Part II

Part II Original Goals for Sprint 2

Original Goals for Sprint 2

Client side

- Build physic engine for Ping-Pong game on stage.js;
- Build rendering logic on client side on stage.js.

Server side

- Build game caching on top of Redis for quick game data retrieve;
- Build synchronous packet forwarding based on channel.

Part III

Part III What we have done

- Requirement
- Built game coordinating logic

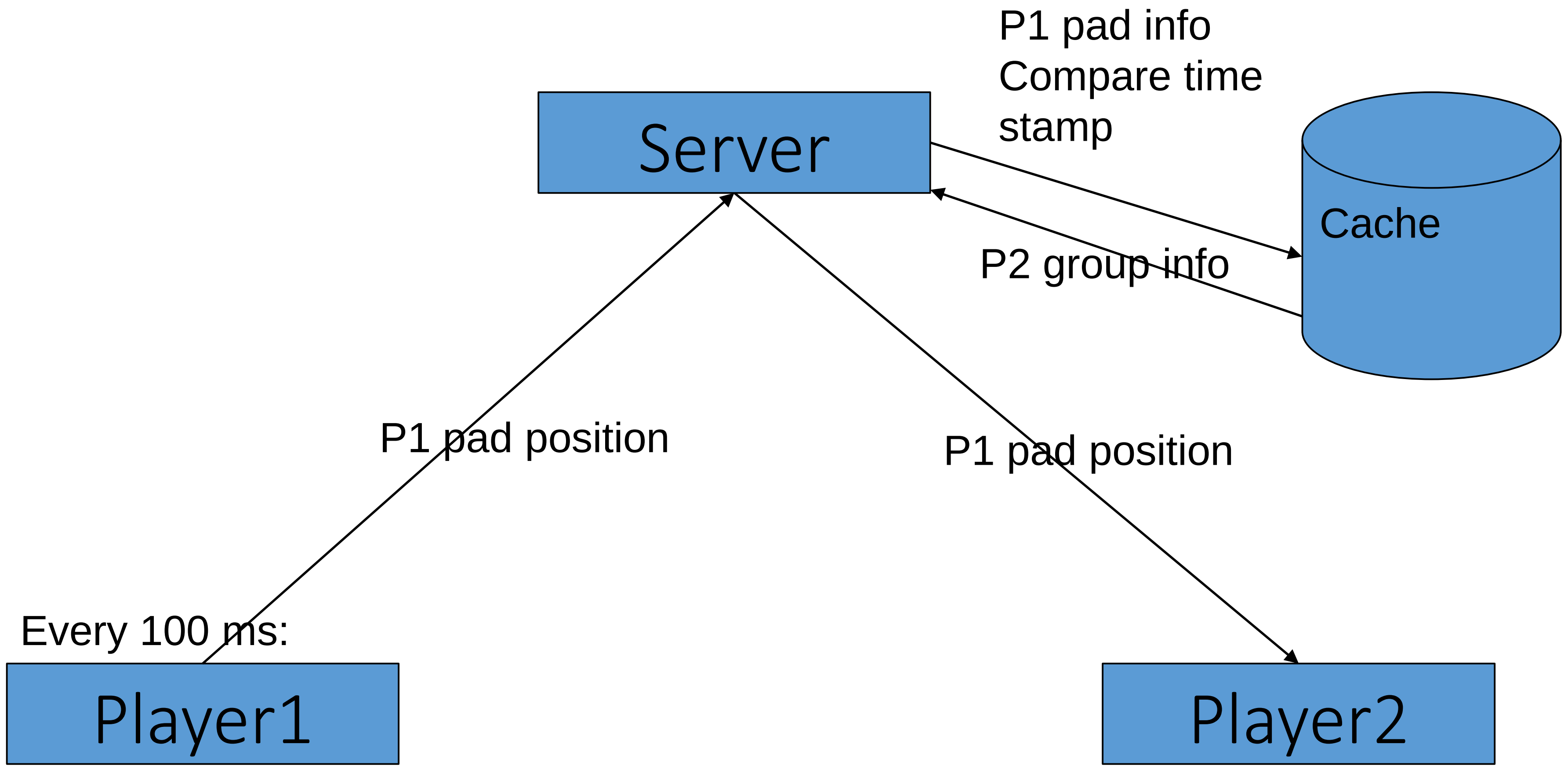
What we have done

Requirement:

- Players want to have smooth control over its pad and ball bouncing;
  - Each client has its own physical engine;
  - Just send pad info periodically;

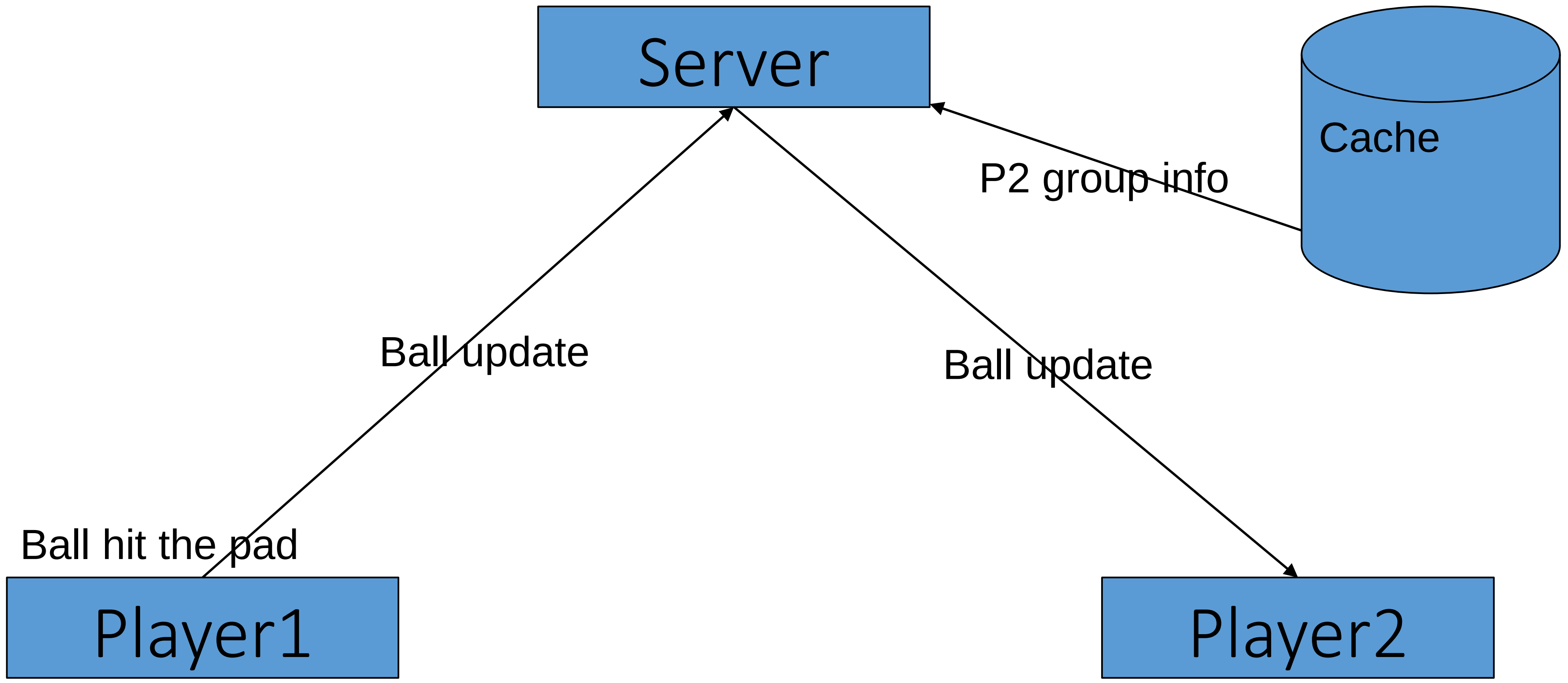
- Player wants to see each other and know others updates;
  - Server would forward update of another players pad
  - Pad hitting ball event would trigger ball info forward;

Pad update

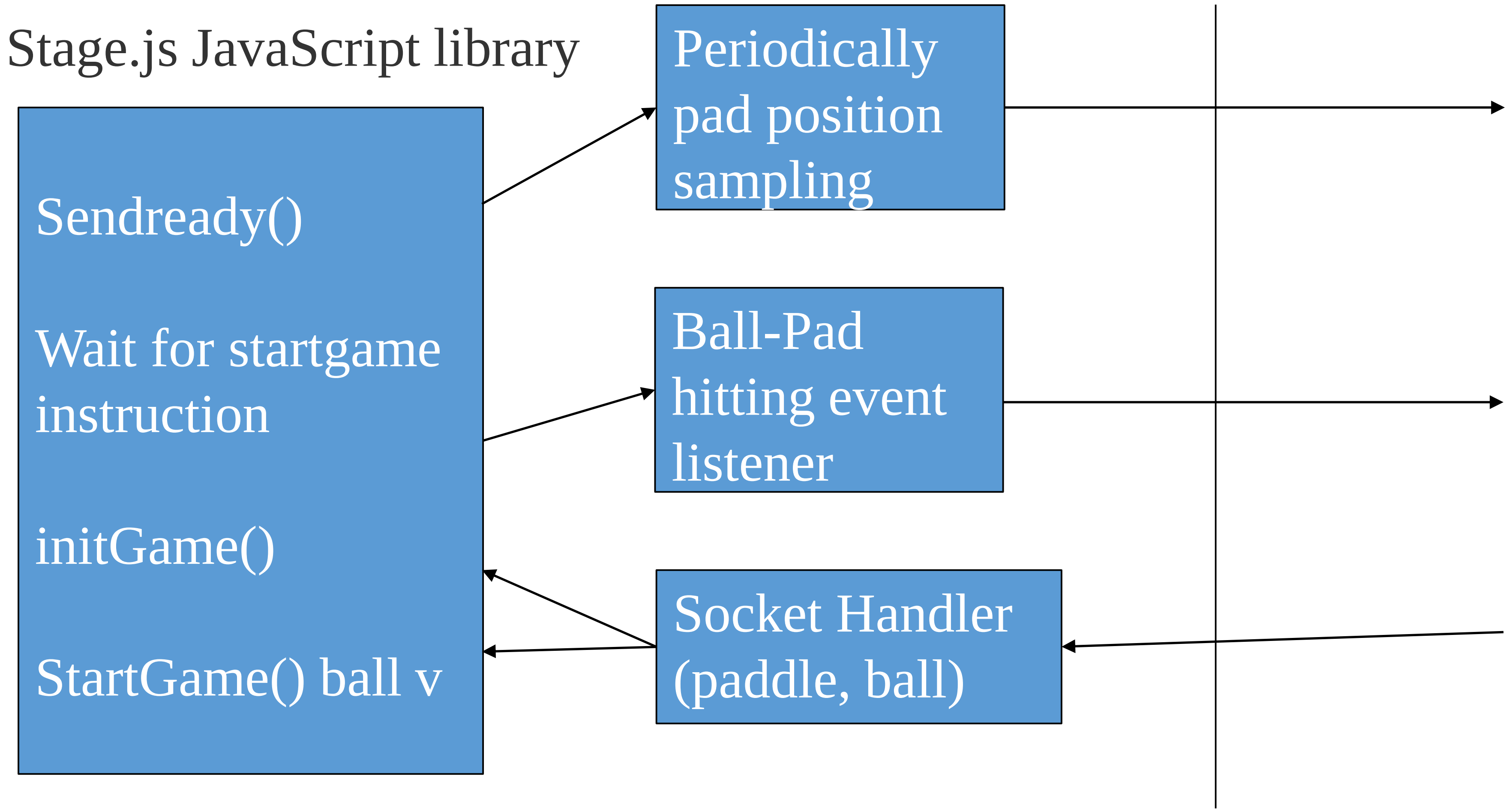


P1 pad info
Compare time stamp
Server
Cache
P2 group info
P1 pad position
P1 pad position
Every 100 ms:
Player1
Player2

Ball Update


Server
Cache
P2 group info
Ball update
Ball update
Ball hit the pad
Player1
Player2

Client Side Logic


Stage.js JavaScript library
Sendready()
Wait for startgame instruction
initGame()
StartGame() ball v
Periodically pad position sampling
Ball-Pad hitting event listener
Socket Handler (paddle, ball)

What we have done

Demonstration

Gaming

- Ball and pads bouncing
- Mouse controlled pad movement

Synchronization

- Pad movement can be seen on the other side
- Ball state would be synchronized

Part IV

Part IV Problems encountered

What we have done

Problem:

- Channel is stateless, worse: no guarantee for order of reply;
- Database access is not fast, we need cache like Redis. However, Redis provide almost no transactional control

Part V

Part V Goals for next sprint

- Client side smoothing
- Deployment and Tuning
- Score Logic

Goals for next sprint

Client side smoothing

- Store the previous snapshot of pad and only render after the next state has come
- Fix client side jittering

Deployment and tuning

- Choose appropriate service to maximize Redis throughput
- Parameter tuning: client side sending rate, channel worker number

Score Logic Testing

- We have implement the state on server side;
- Client side need to implement state handler logic.

Thank you!