Today

Midterm: Oct 5

Synch + Comm. in Parallel systems

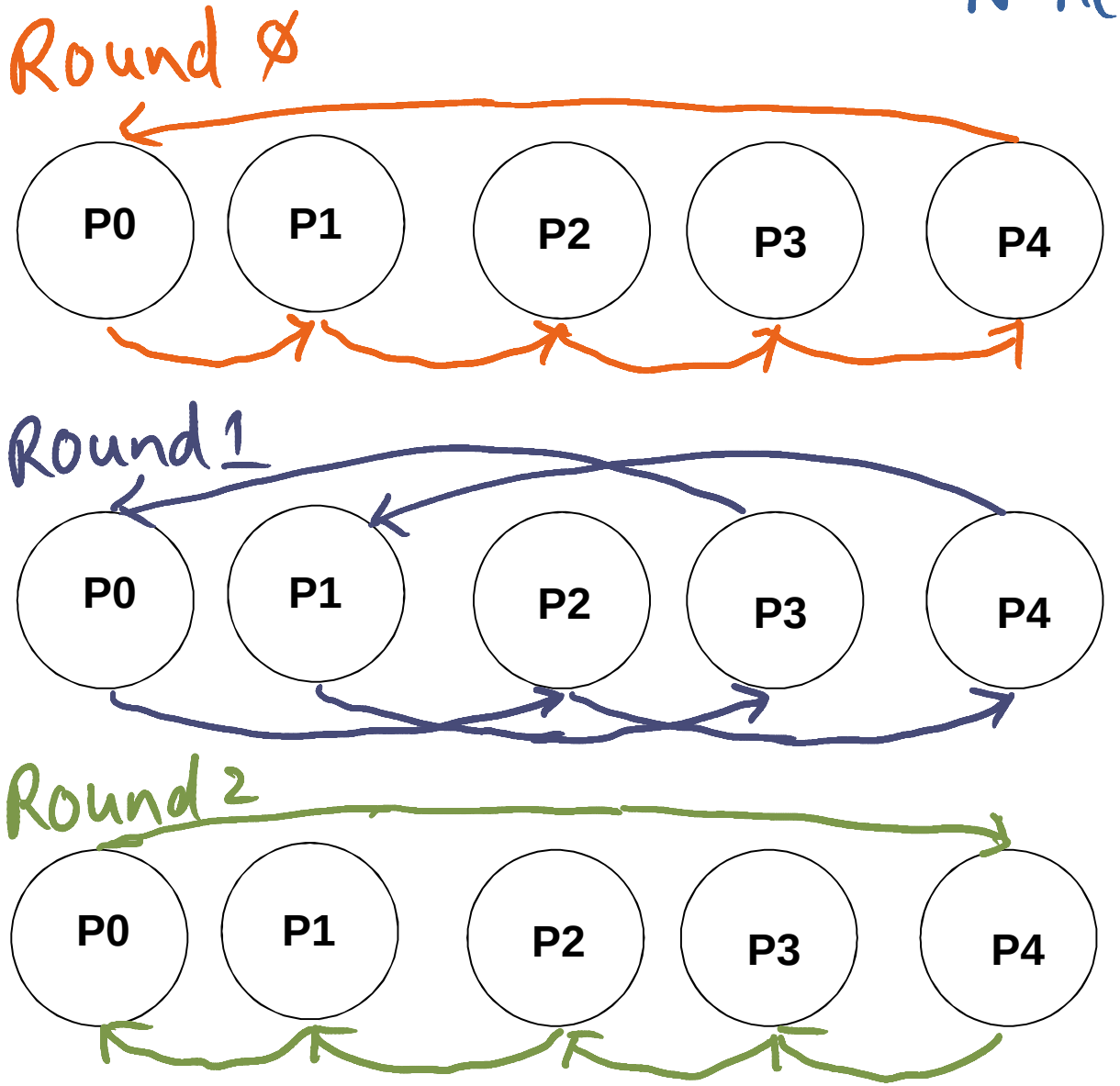
* MCS paper
  - ✓ * Parallel systems basics
  - ✓ * Spin lock algorithms
  - ✓ * Barrier algorithms
    - ✓ - Counting, Sense, tree, MCS
    - ✓ - tournament, dissemination

⇒ * Cross domain communication (CRPC)

Wednesday

* Sched in SM systems

Dissemination Barrier



N need not be power of 2

Each round(k):

$P_i \rightarrow P_{(i+2^k) \bmod N}$

O(N) comm. events per round

- Ceil($\log_2 N$) rounds

All Awake !

- No hierarchy
- Works for Ncc and clusters

Virtues of Dissemination

- No hierarchy
- No pair wise sync
- Each node works independently to send out the messages per protocol
- All will realize the barrier is complete when they have received the ceil(log2N) messages from their peers
- Total amount of comm: $O(N \log_2 N)$
- In comparison, tournament incurs $O(\log_2 N)$ comm since in that set up the comm decreases as we go up the tree; since dissemination is based on gossip, the amount of comm in each round is constant and equal to N
- Works for NCC and MP machines in addition to SM

Performance Evaluation

Performance Evaluation

Spin Algorithms →
- Spin on T+S
- Spin on read
- Spin with delay
- Ticket lock
- Array queue lock
- List queue lock

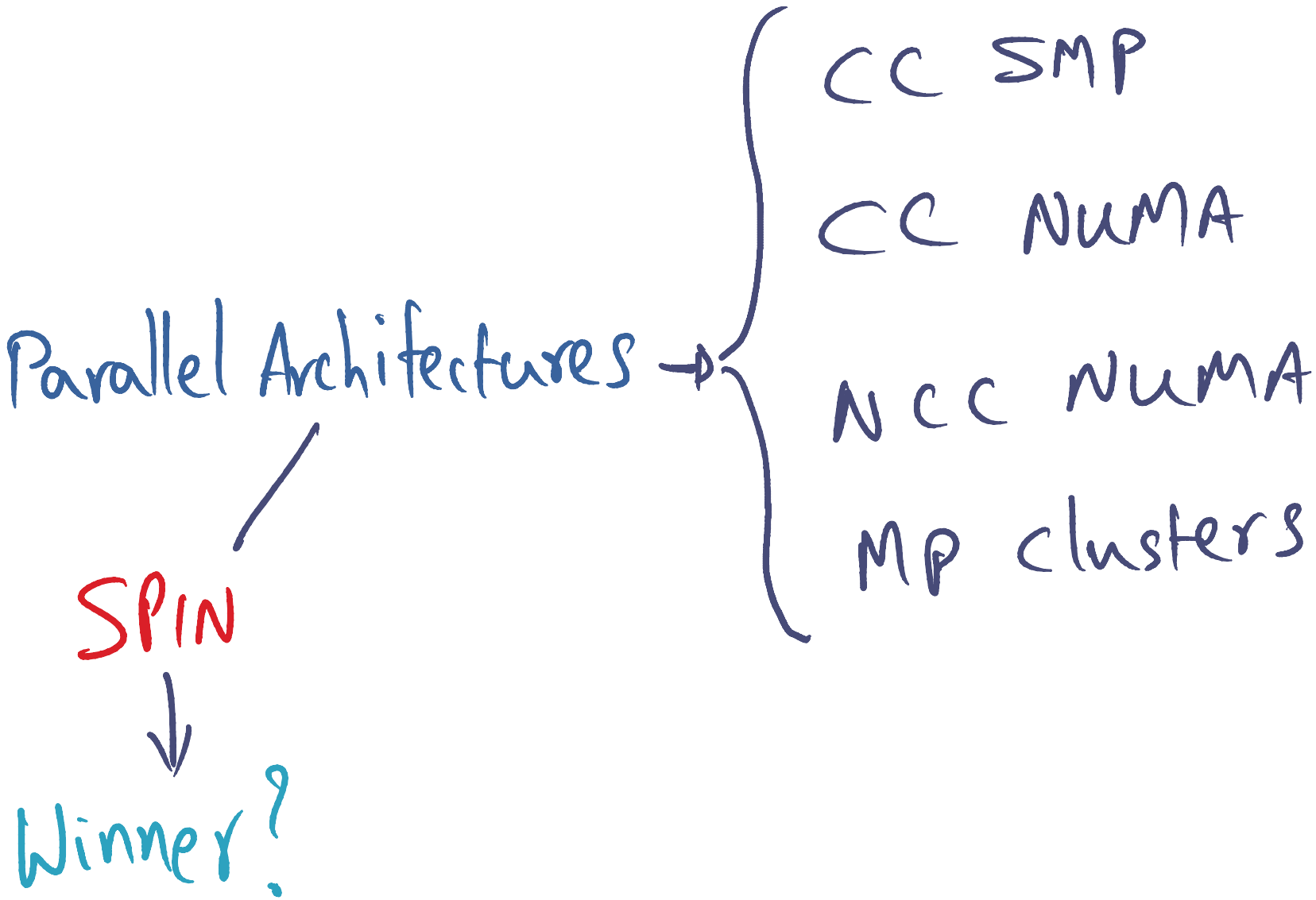
Performance Evaluation

Barrier Algorithms →
- Counter
- Tree
- MCS Tree
- Tournament
- Dissemination

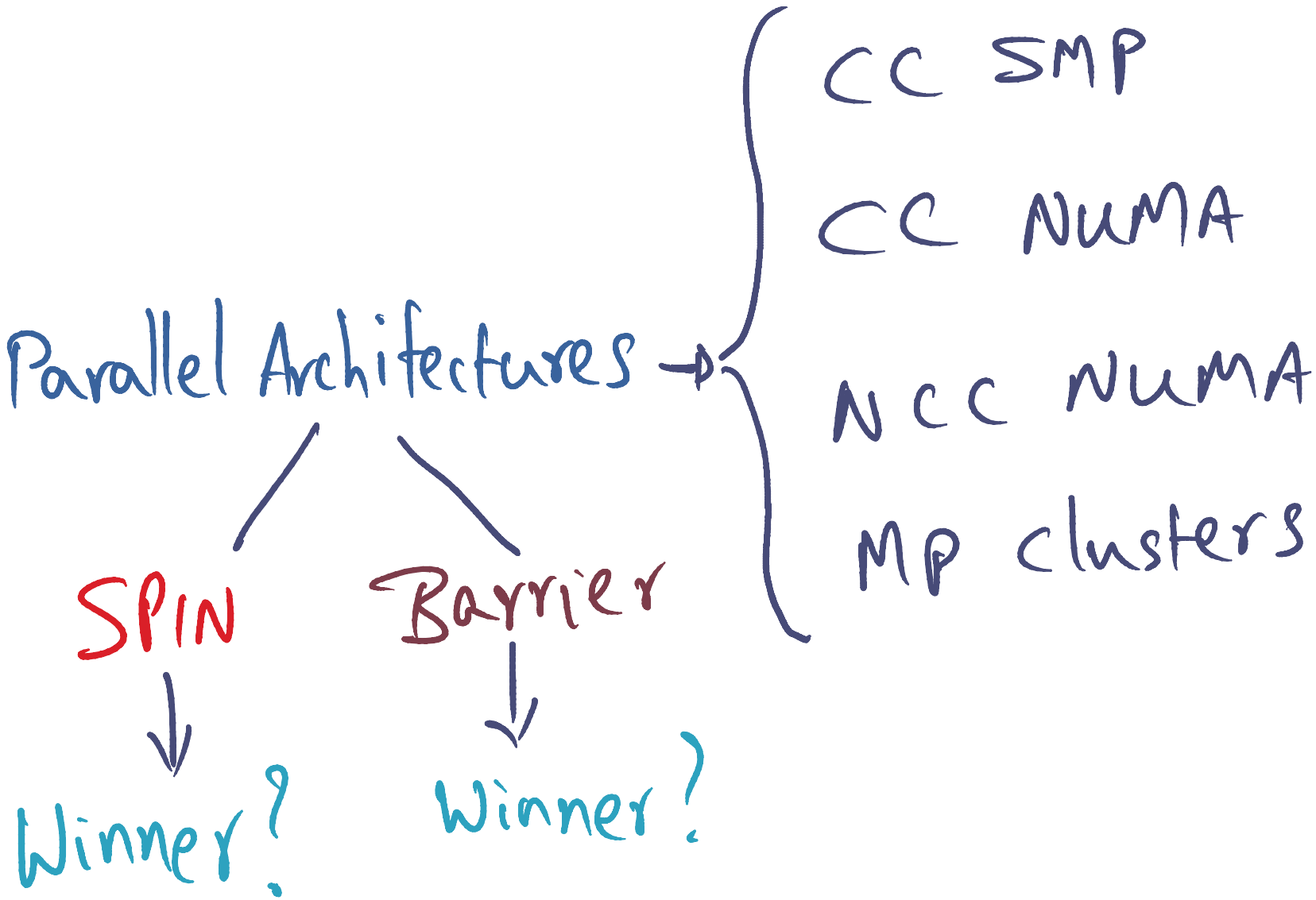
Performance Evaluation

Parallel Architectures →
- CC SMP
- CC NUMA
- NCC NUMA
- MP clusters

Performance Evaluation



Performance Evaluation



BBN Butterfly

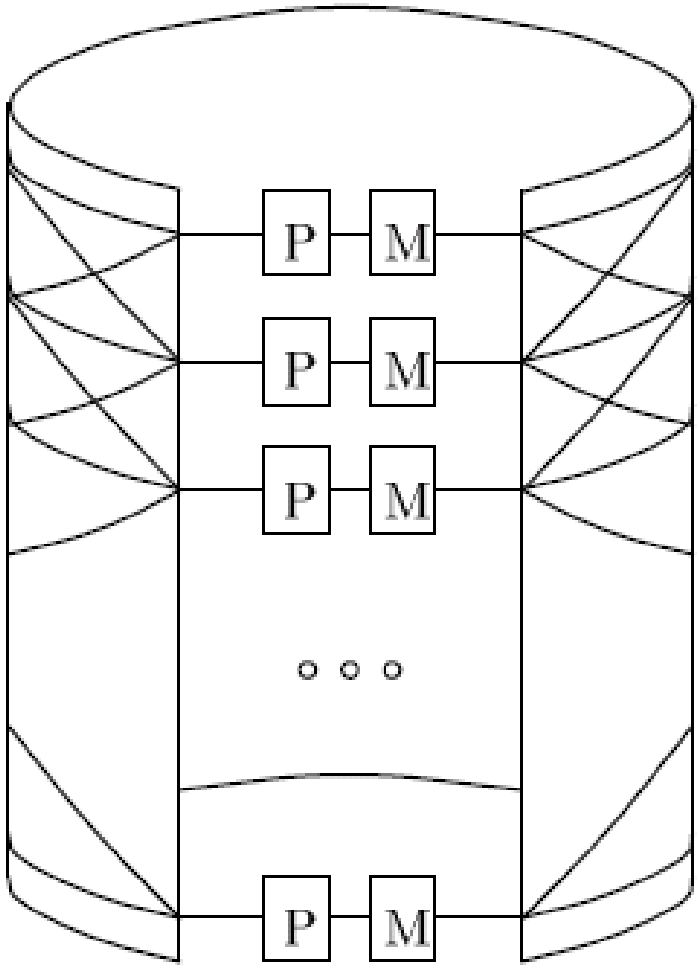


Figure 2: The BBN Butterfly 1.

Sequent Symmetry

CC - SMP with shared bus or ICN

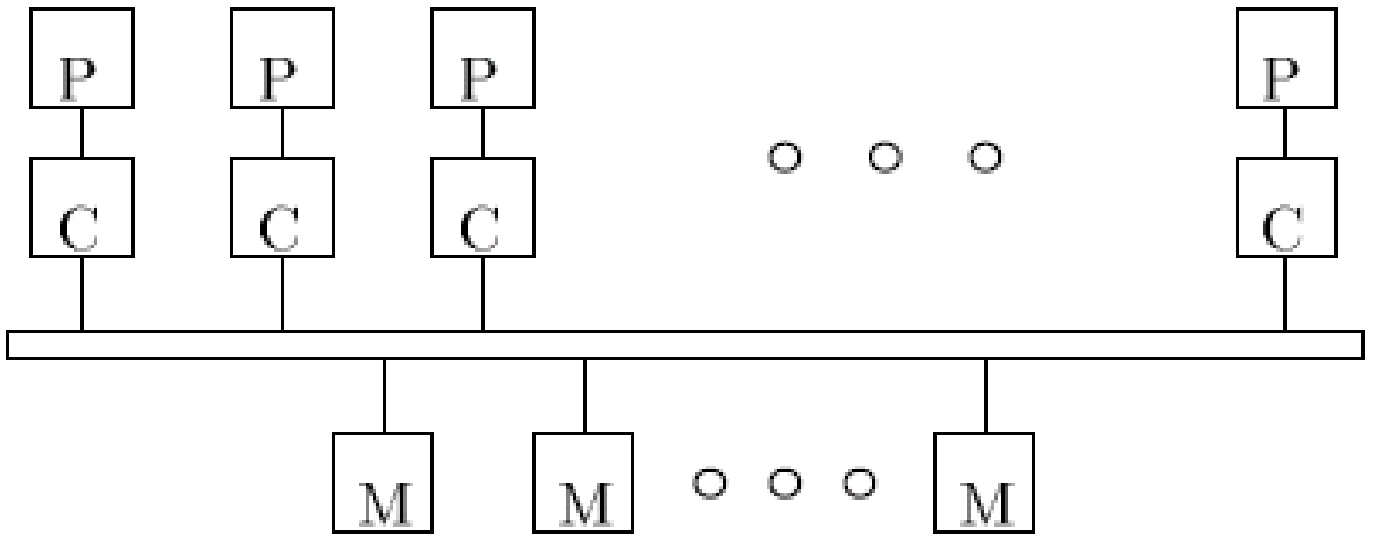


Figure 3: The Sequent Symmetry Model B.

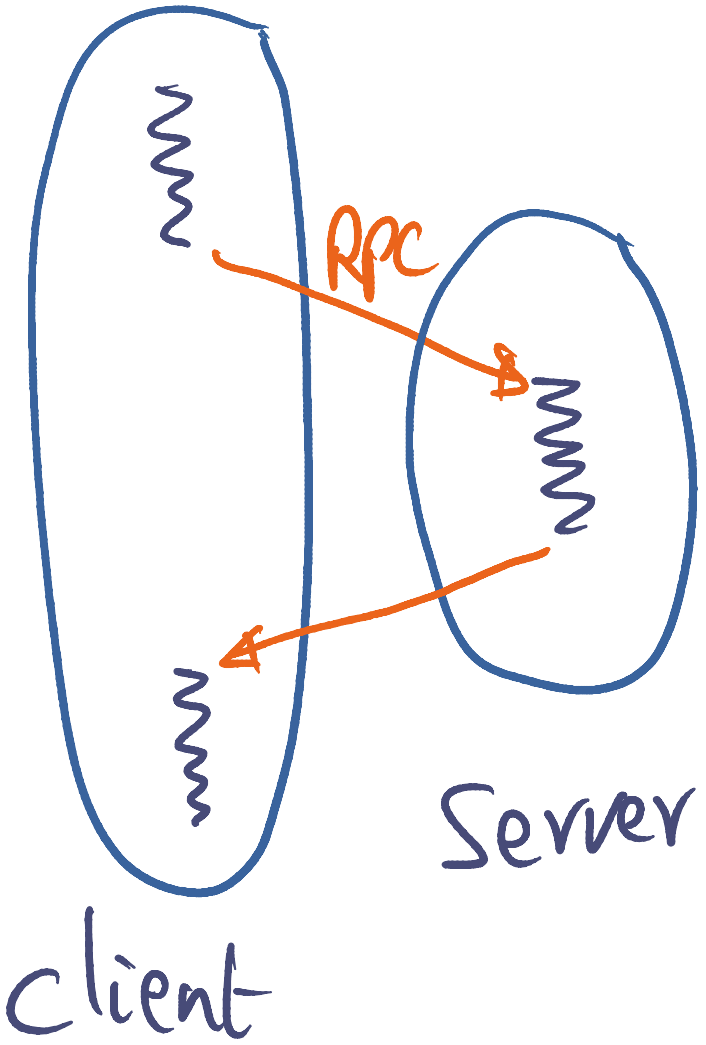
Performance

- form an intuition
- verify whether reported results agree with your opinion

- Spinlock
  - Which one do you think will win on symmetry?
  - Which one do you think will win on BBN?
- Barrier
  - Which one do you think will win on symmetry?
  - Which one do you think will win on BBN?

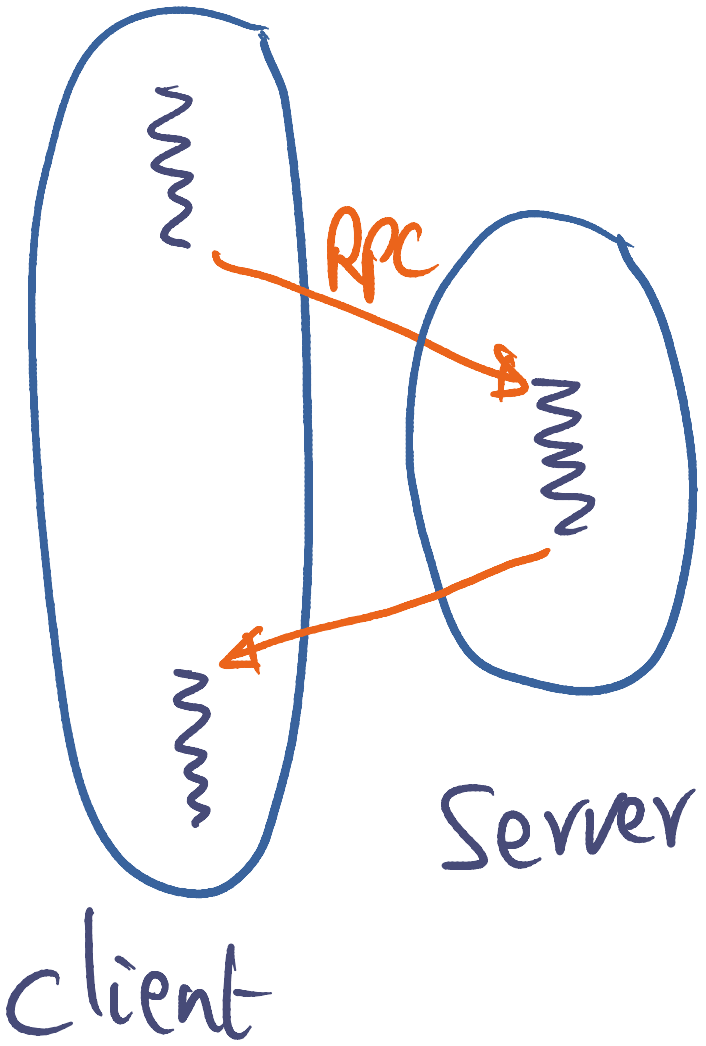
Compare their barrier and lock results on symmetry and BBN against the results reported for the same algorithms on a different parallel machine, namely, KSR-1 in the paper "Scalability study of KSR-1"
(See T-Square Modules: Section 10.10)

RPC and client-server Systems

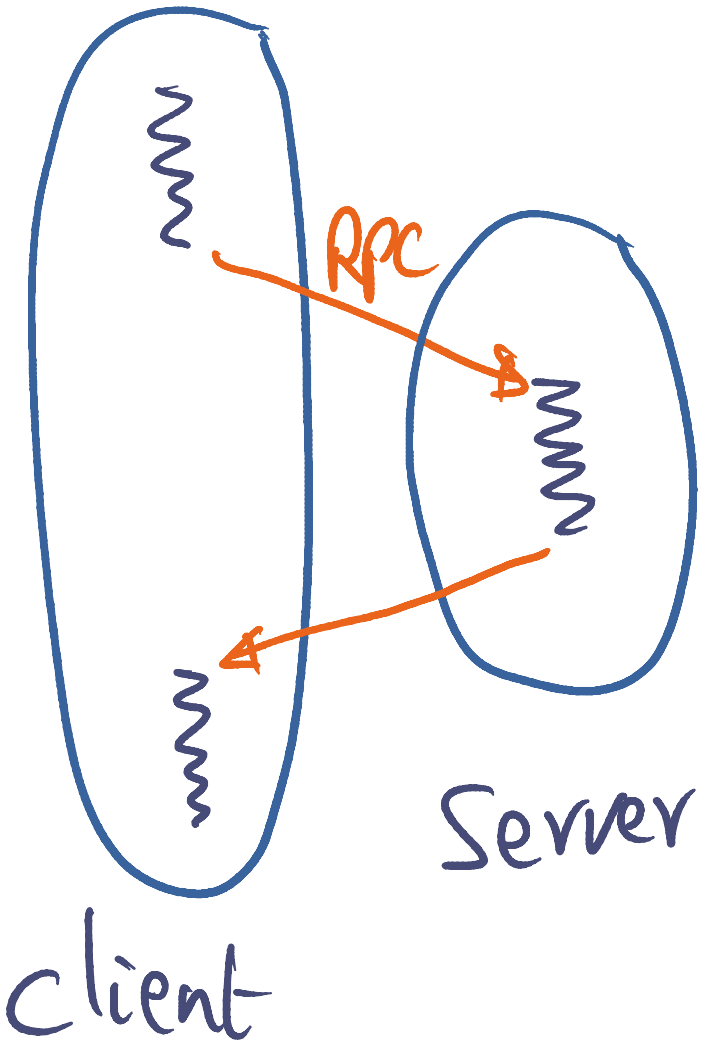


RPC and client-server Systems

E.g. File System



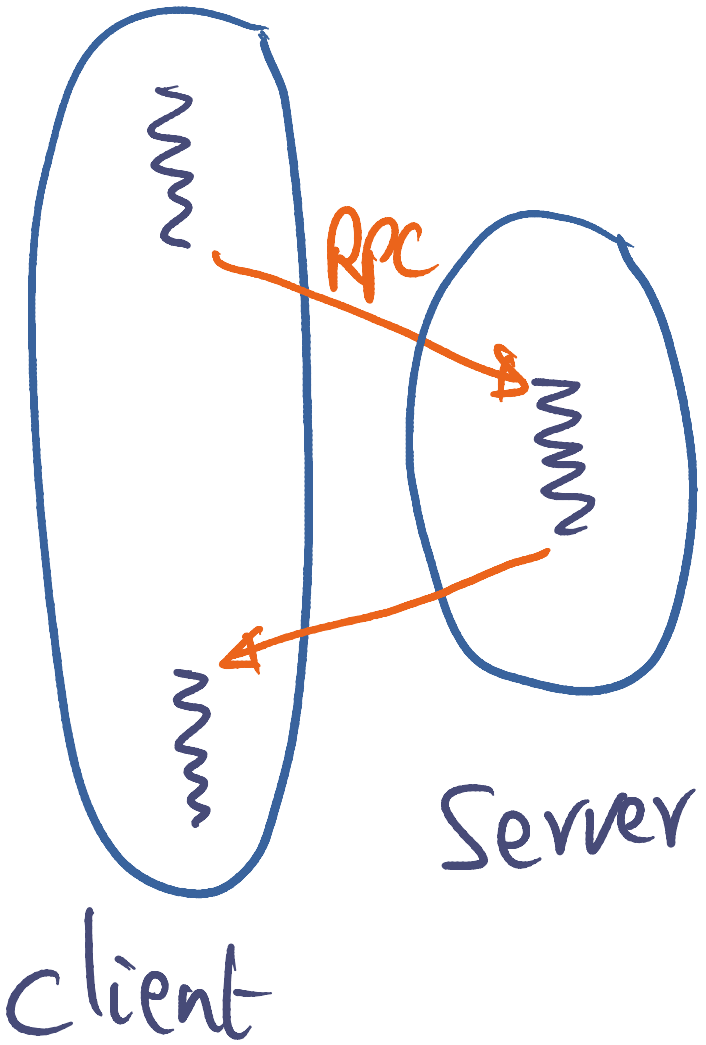
RPC and client-server Systems



E.g. File System

RPC: usually remote

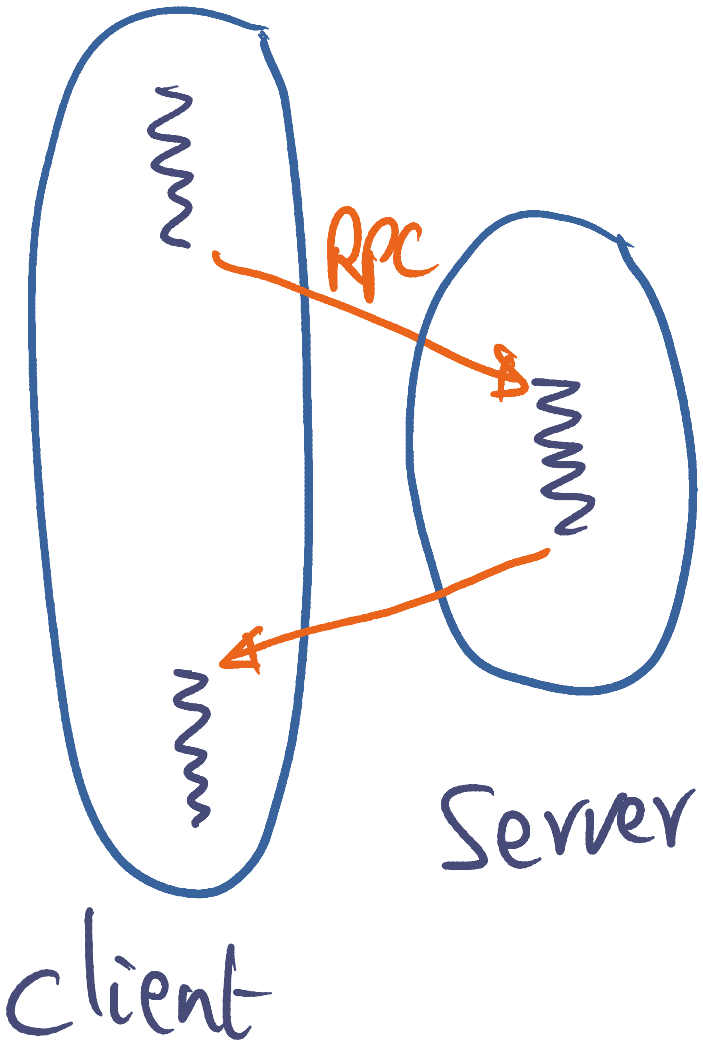
RPC and client-server Systems



E.g. File System

RPC: usually remote client-server on same machine?

RPC and client-server Systems



E.g. File System

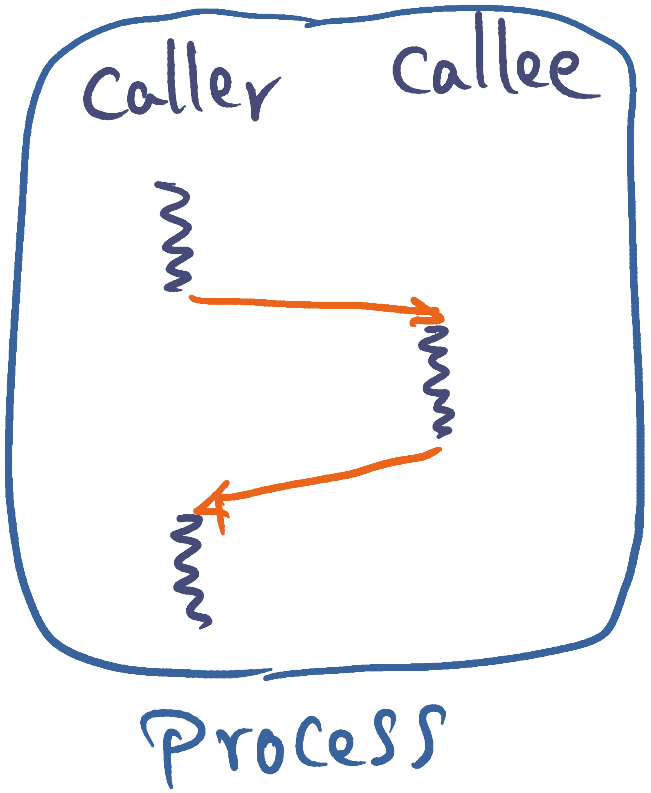
RPC: usually remote

client-server on same machine?

- Performance Vs. Safety

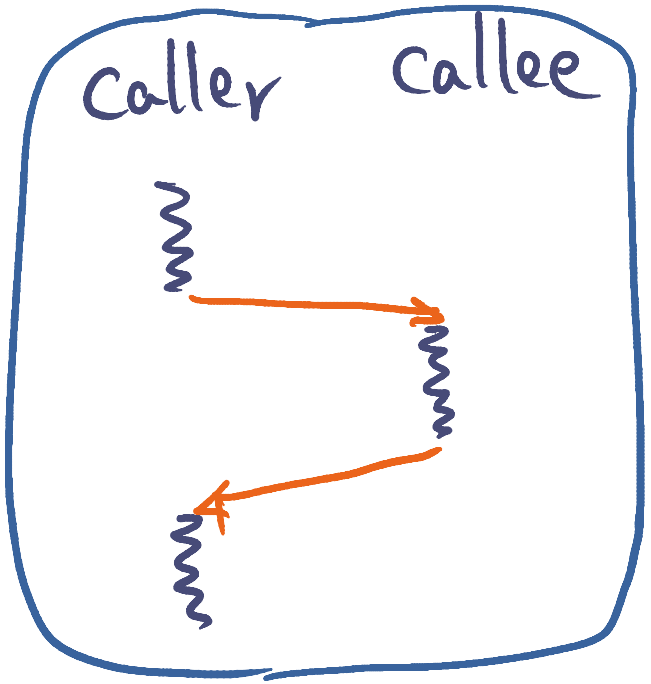
RPC Vs. Simple Procedure Call

Procedure Call



RPC Vs. Simple Procedure Call

Procedure Call

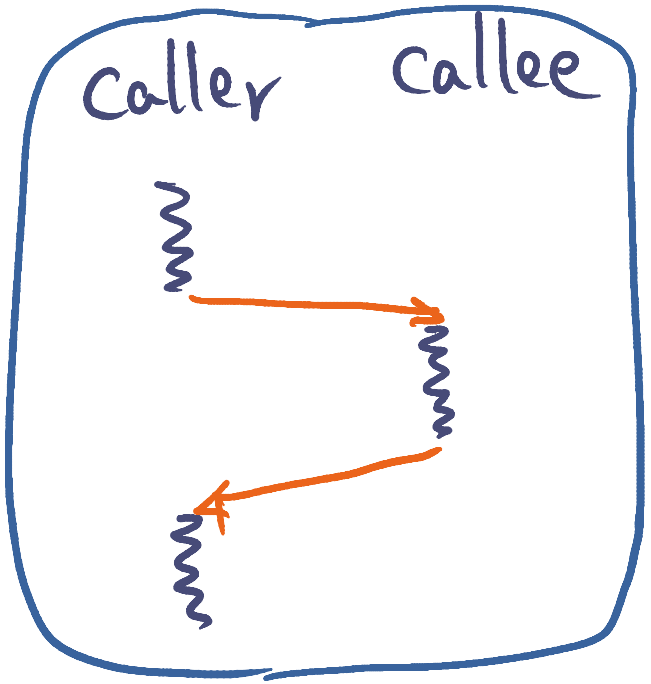


Process

- All at Compiletime

RPC Vs. Simple Procedure Call

Procedure Call

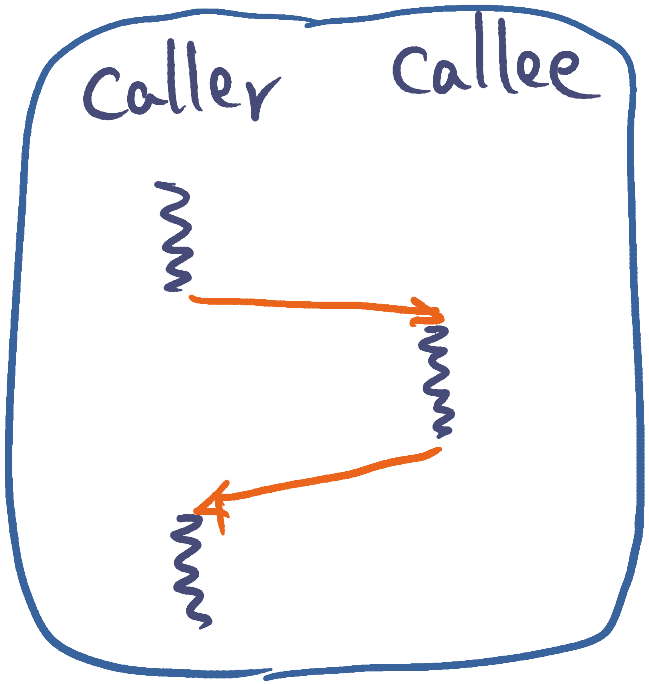


Process

- All at Compiletime

RPC Vs. Simple Procedure Call

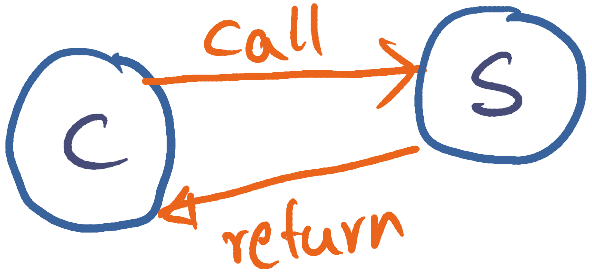
Procedure Call



Process

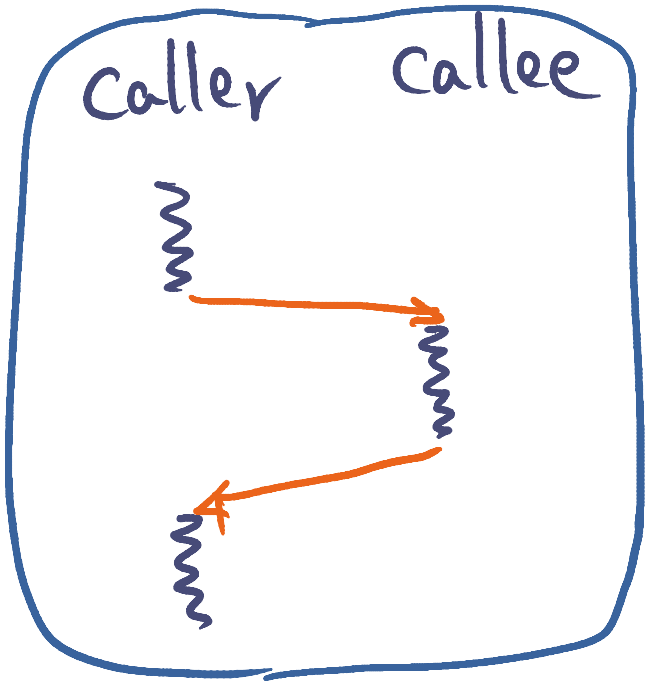
- All at Compiletime

RPC



RPC Vs. Simple Procedure Call

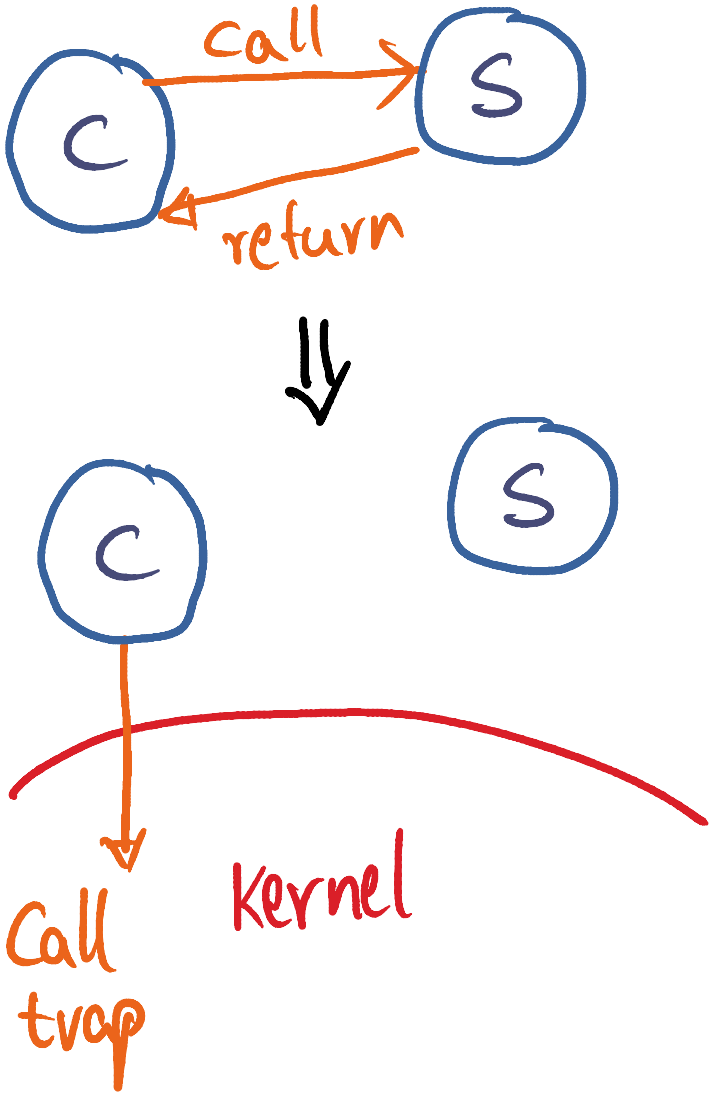
Procedure Call



Process

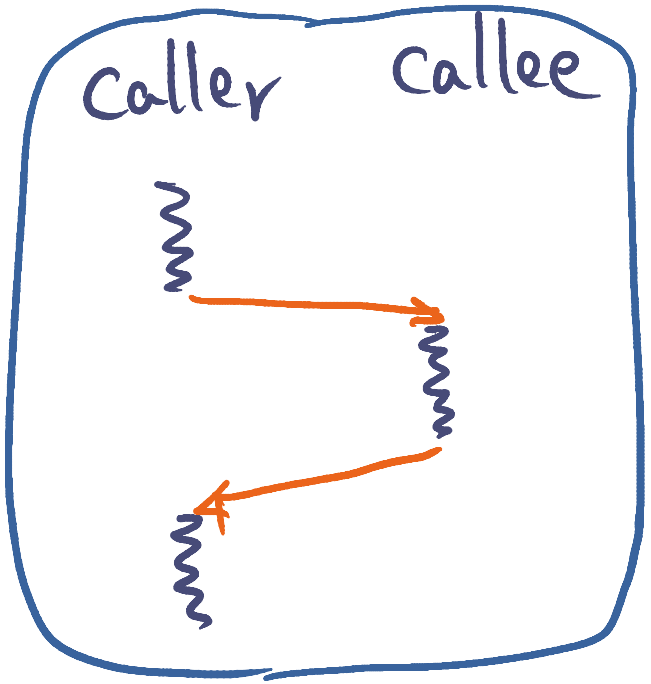
- All at Compiletime

RPC



RPC Vs. Simple Procedure Call

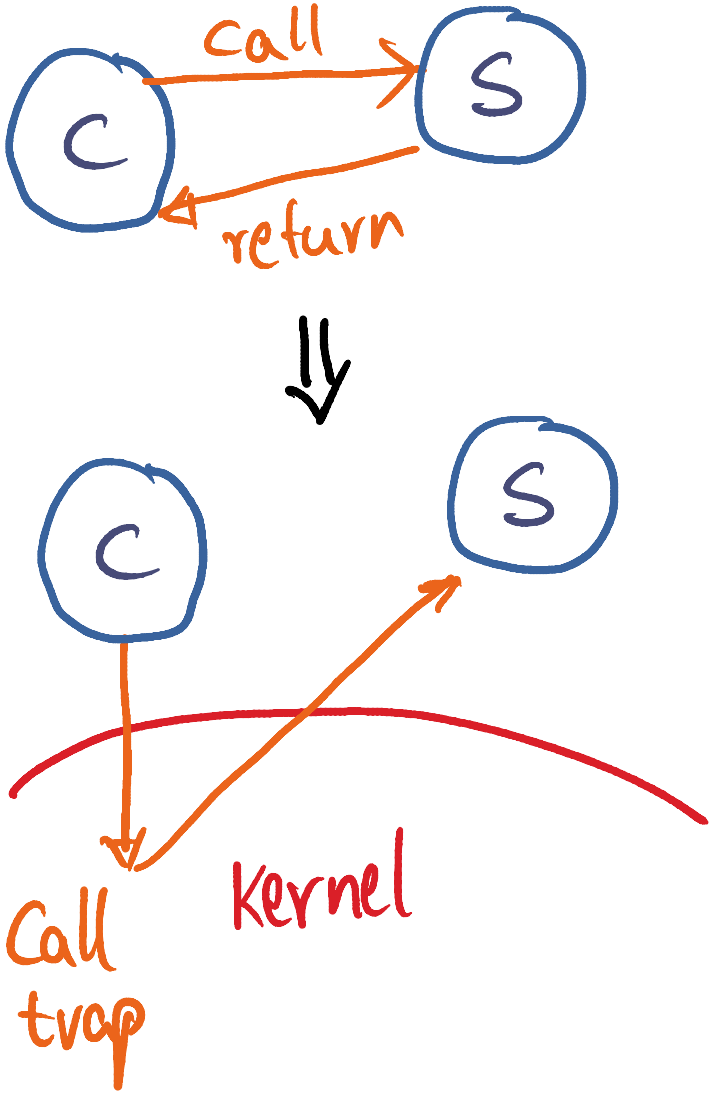
Procedure Call



Process

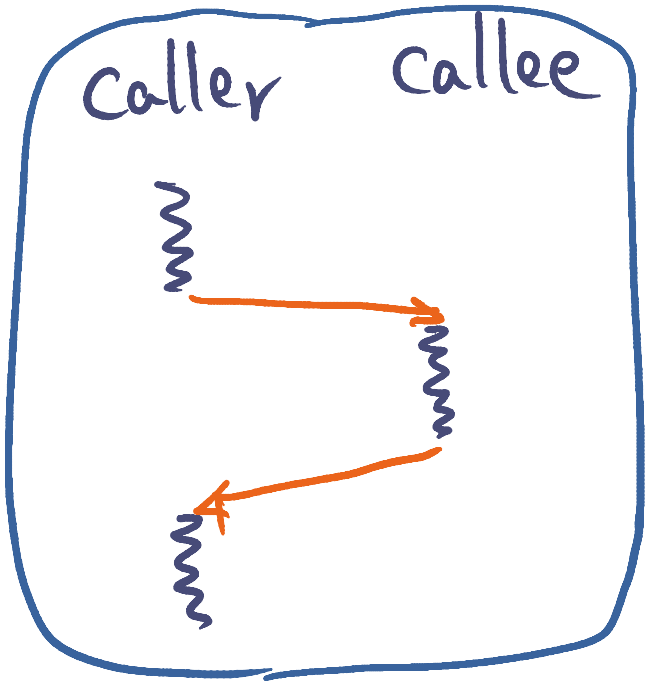
- All at Compiletime

RPC



RPC Vs. Simple Procedure Call

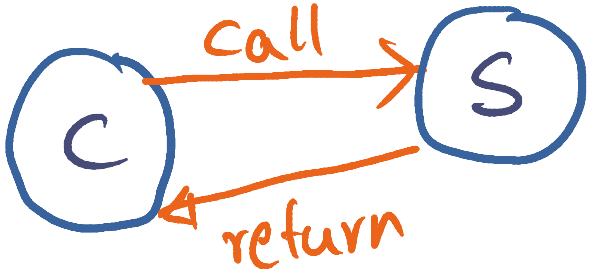
Procedure Call

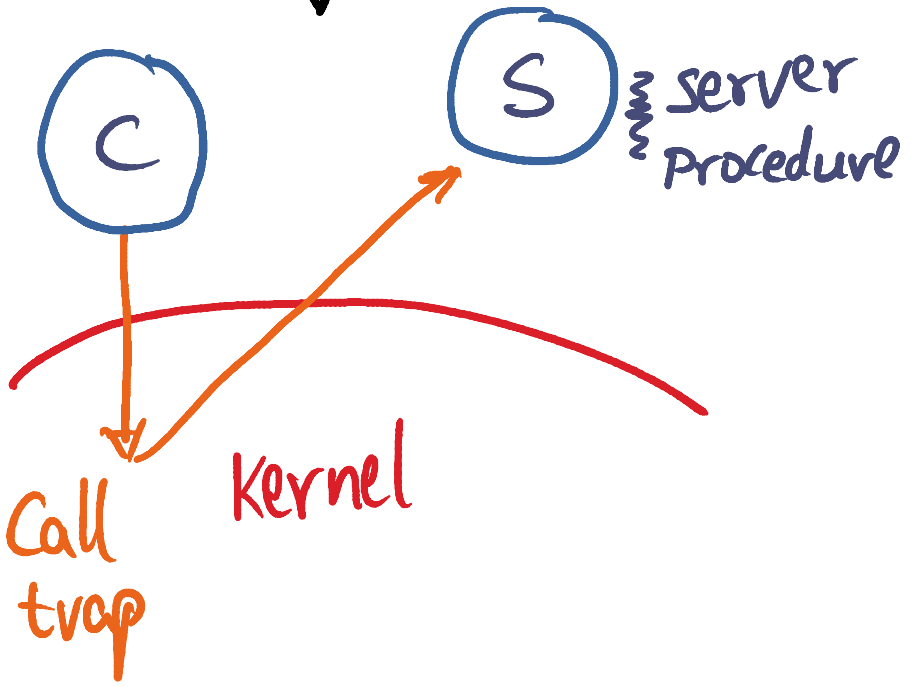


Process

- All at Compiletime

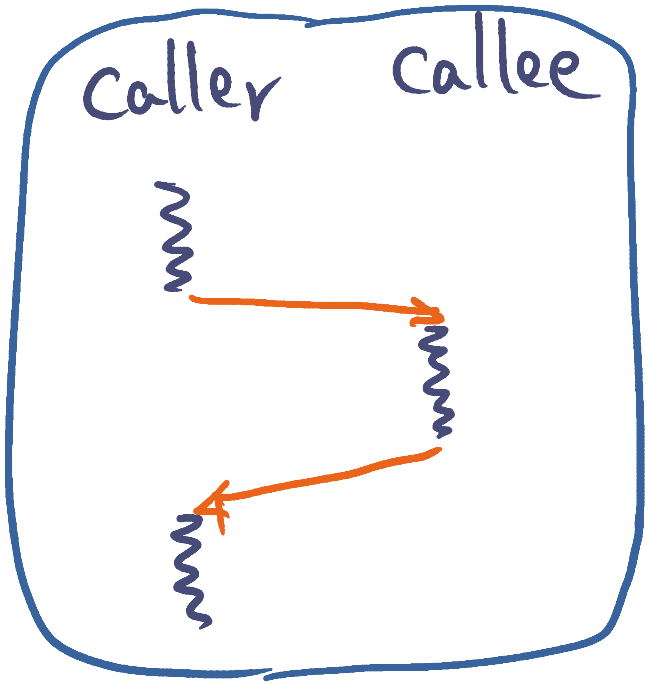
RPC





RPC Vs. Simple Procedure Call

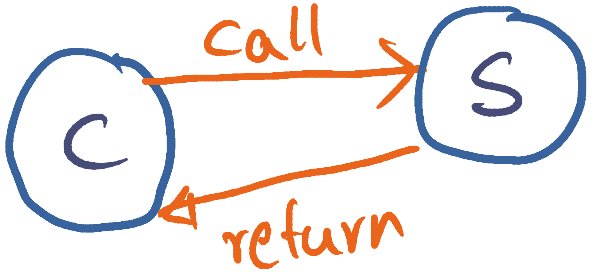
Procedure Call

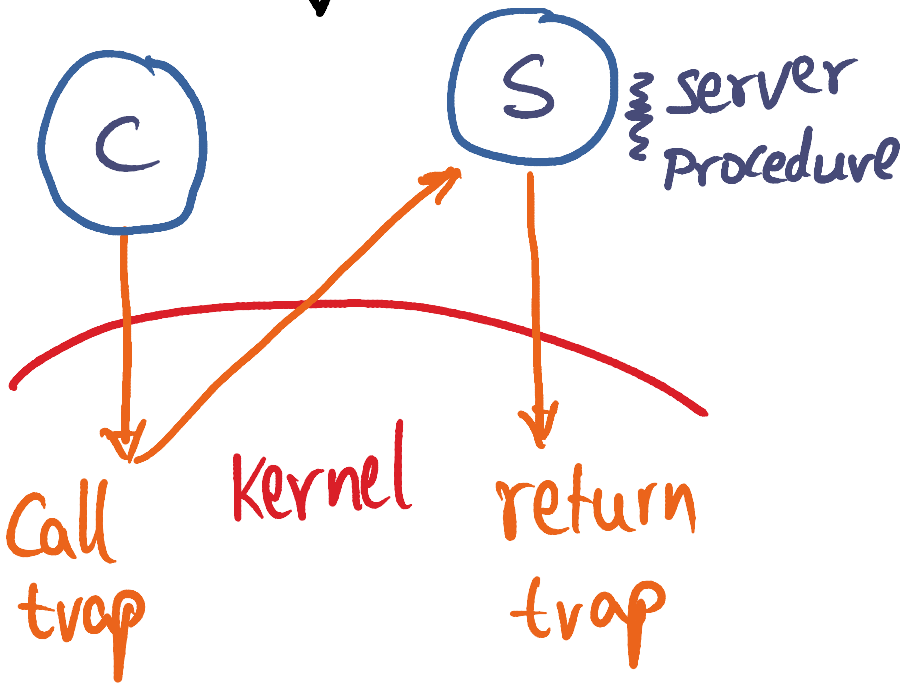


Process

- All at Compiletime

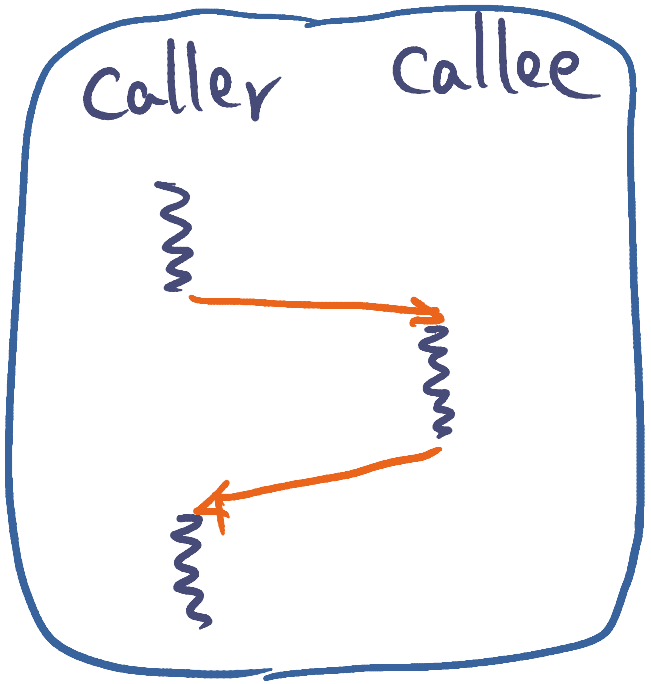
RPC





RPC Vs. Simple Procedure Call

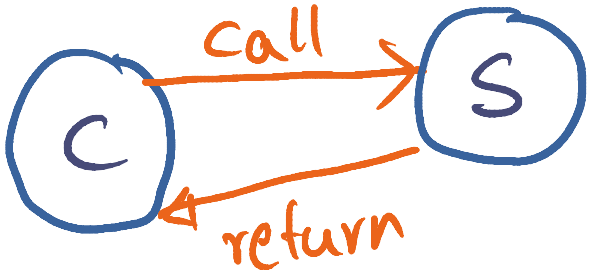
Procedure Call

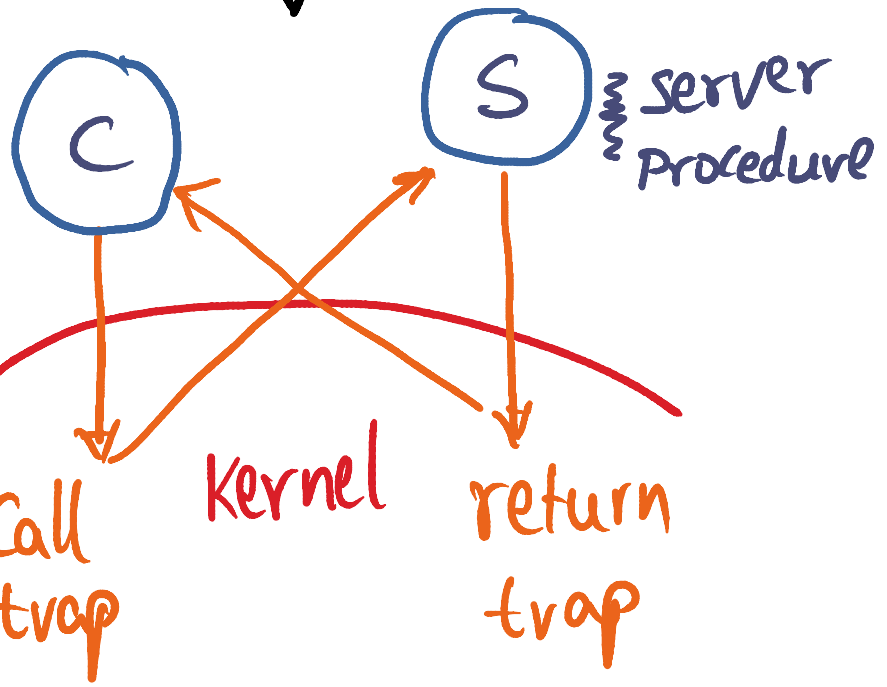


Process

- All at Compiletime

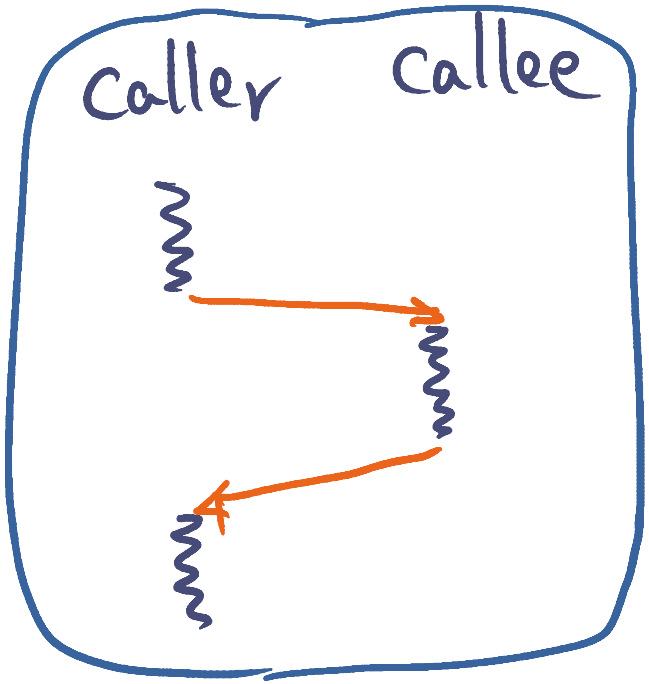
RPC





RPC Vs. Simple Procedure Call

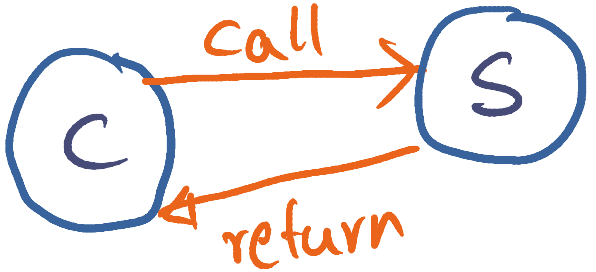
Procedure Call



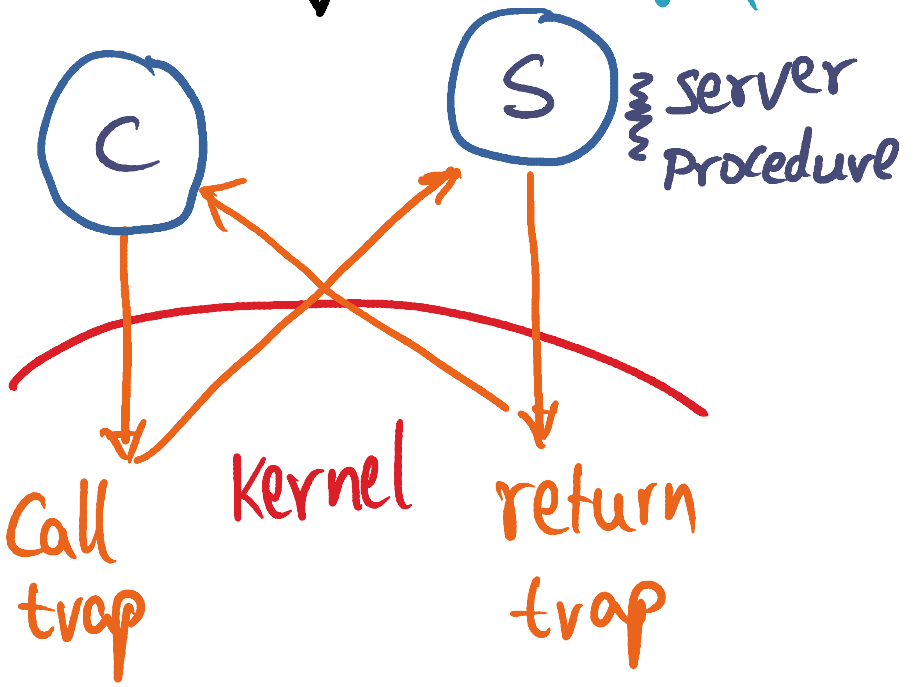
Process

- All at Compiletime

RPC

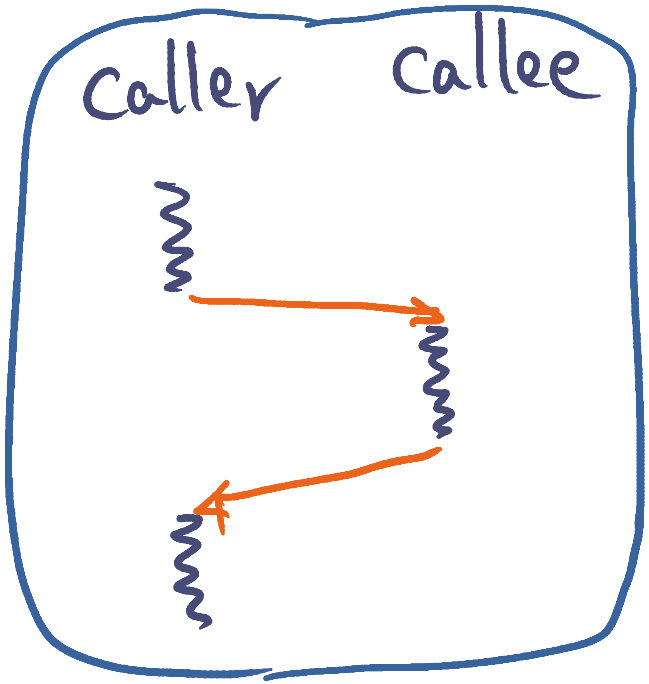


All at runtime



RPC Vs. Simple Procedure Call

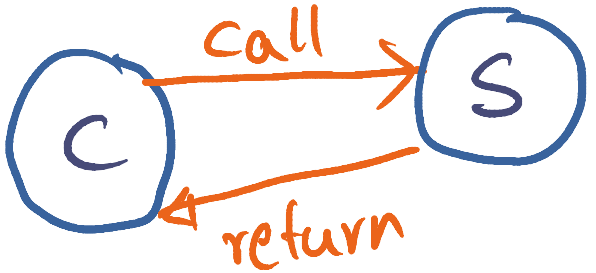
Procedure Call



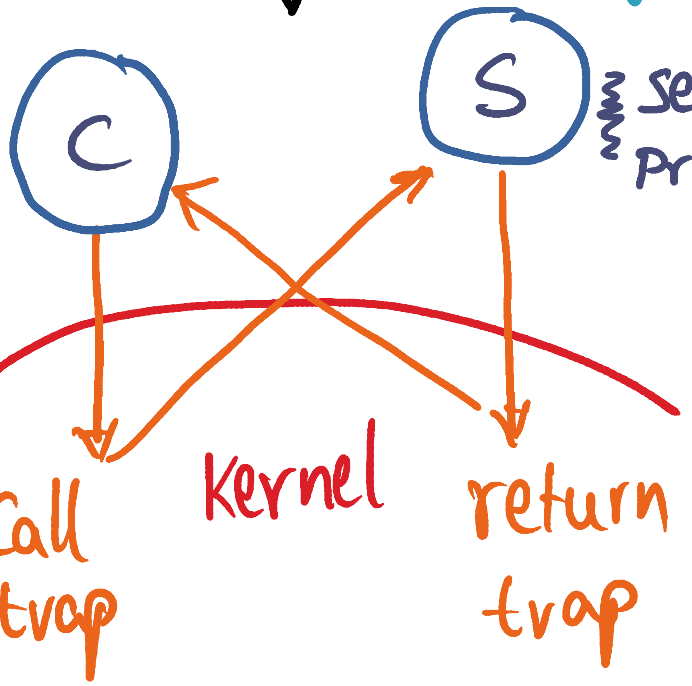
Process

– All at Compiletime

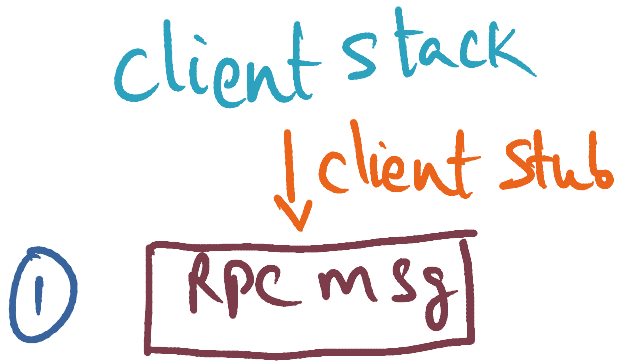
RPC



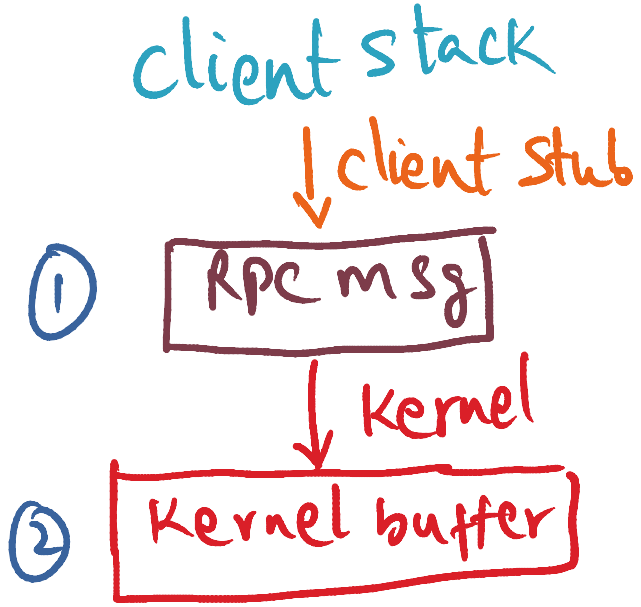
All at runtime



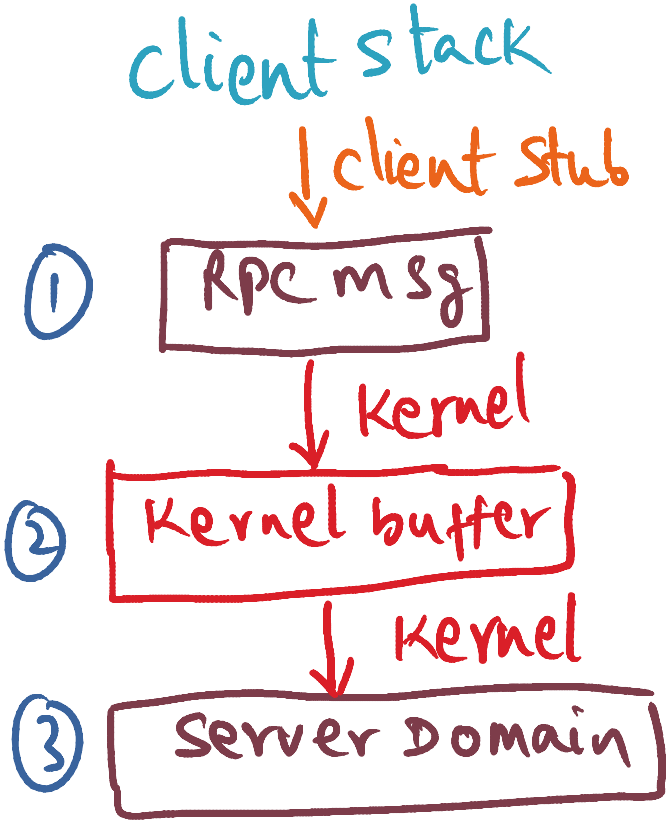
Copying overhead



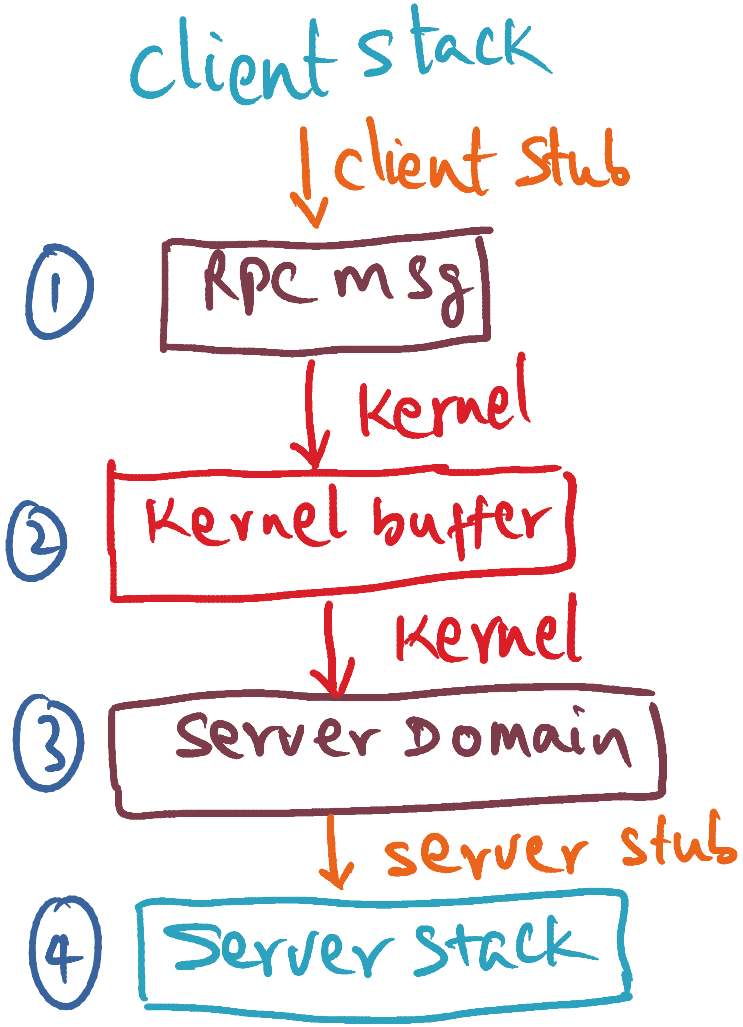
Copying overhead



Copying overhead



Copying overhead



Making RPC cheap

Making RPC cheap

How to remove overheads?

Making RPC cheap (Binding)

How to remove overheads?

Making RPC cheap (Binding)

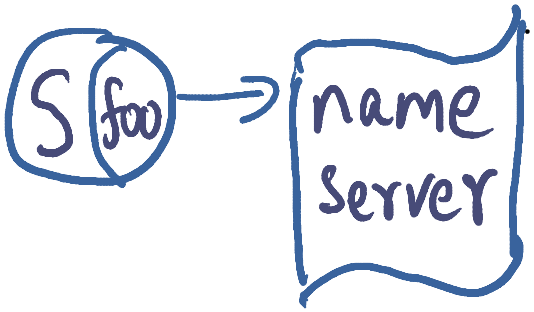
How to remove overheads?

- Set up (Binding) ⇒ one time cost

Making RPC cheap (Binding)

How to remove overheads?

- Set up (Binding) ⇒ one time cost



Making RPC cheap (Binding)

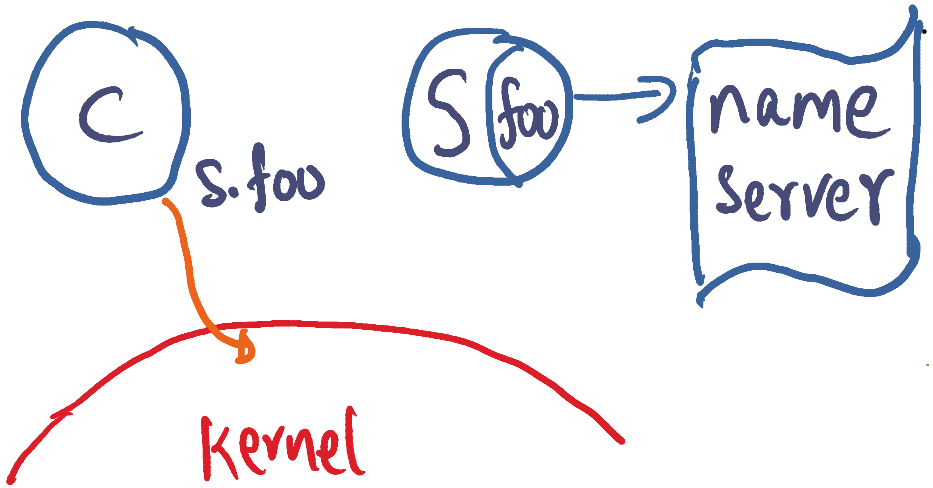
How to remove overheads?

- Set up (Binding) ⇒ one time cost

Making RPC cheap (Binding)

How to remove overheads?

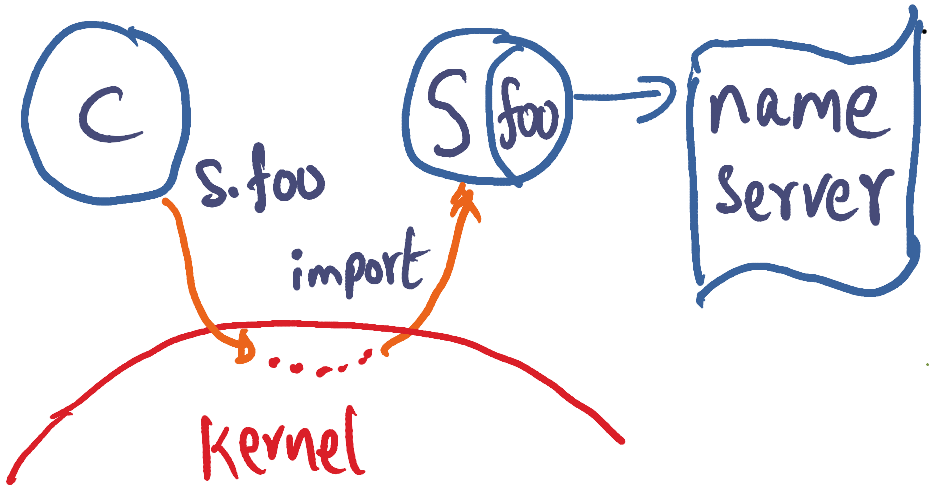
- Set up (Binding) ⇒ one time cost



Making RPC cheap (Binding)

How to remove overheads?

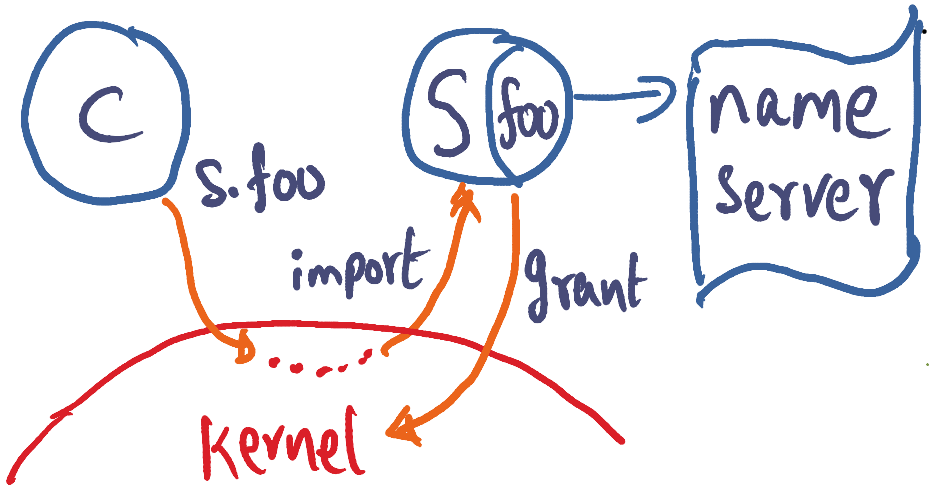
- Set up (Binding) ⇒ one time cost



Making RPC cheap (Binding)

How to remove overheads?

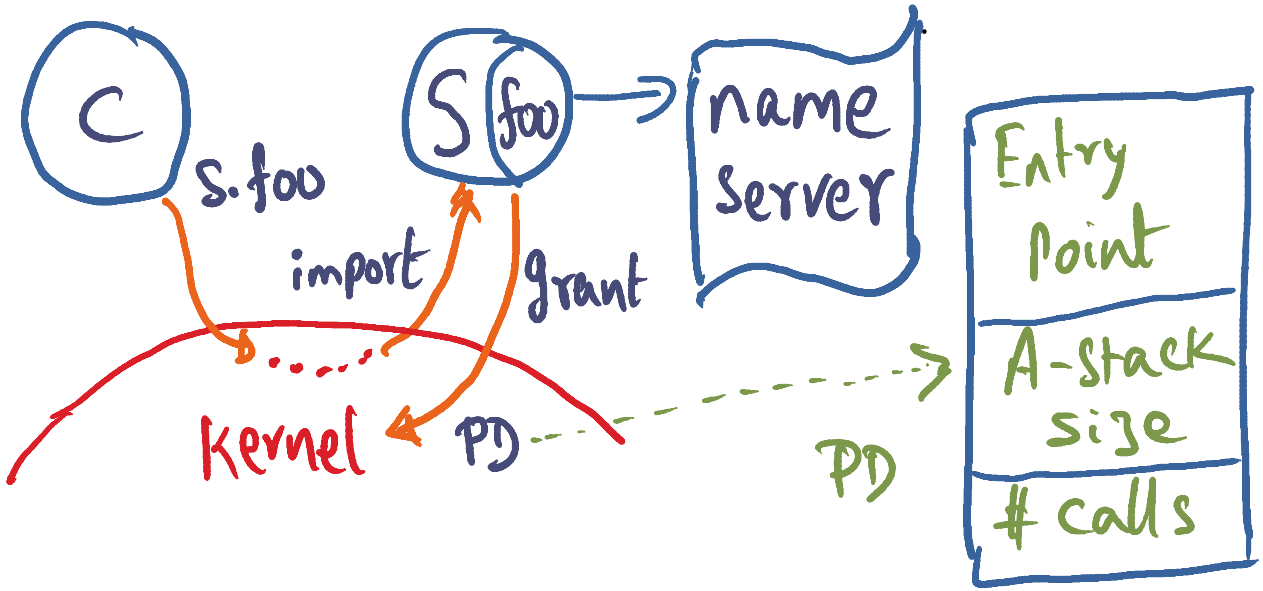
- Set up (Binding) ⇒ one time cost



Making RPC cheap (Binding)

How to remove overheads?

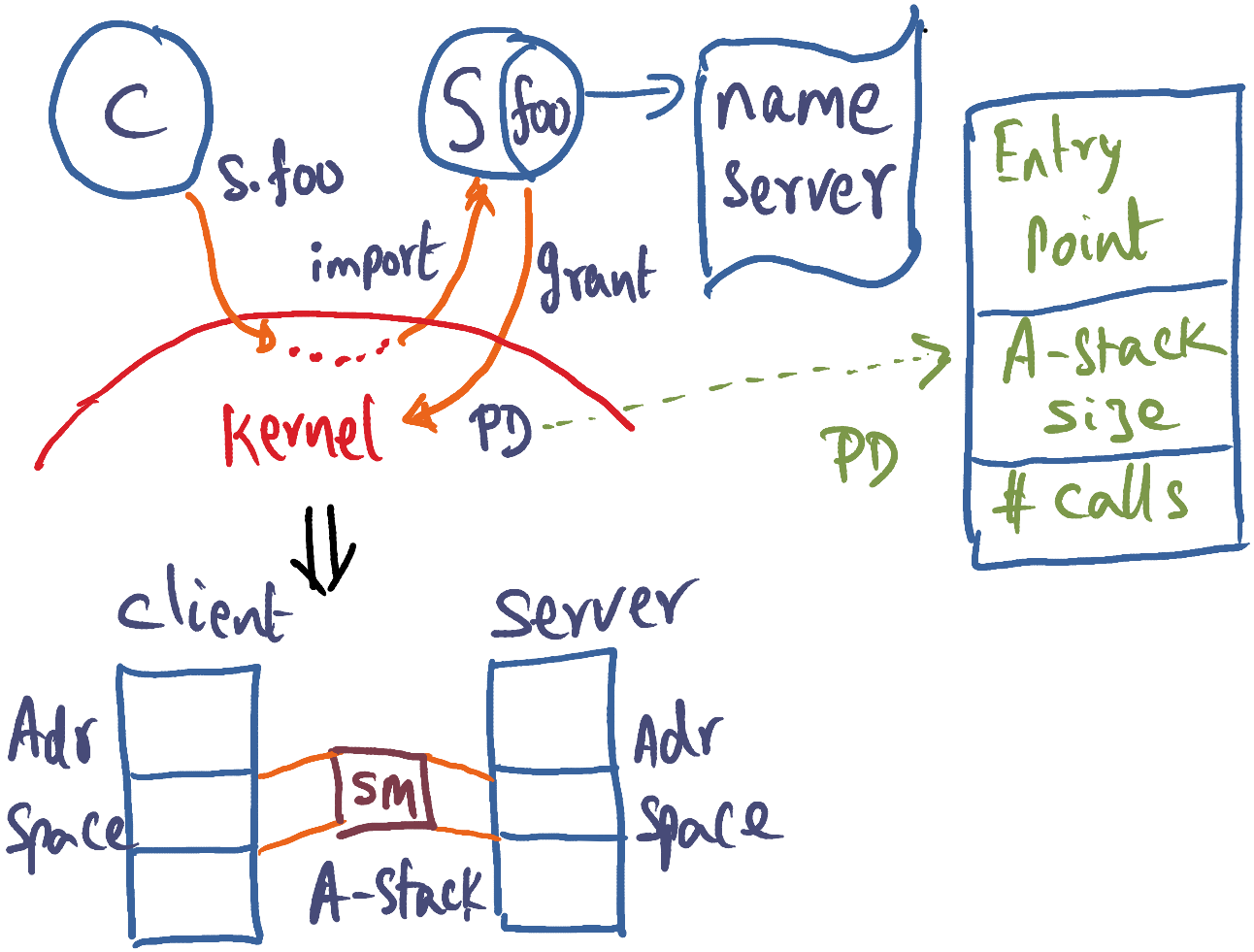
- Set up (Binding) ⇒ one time cost



Making RPC cheap (Binding)

How to remove overheads?

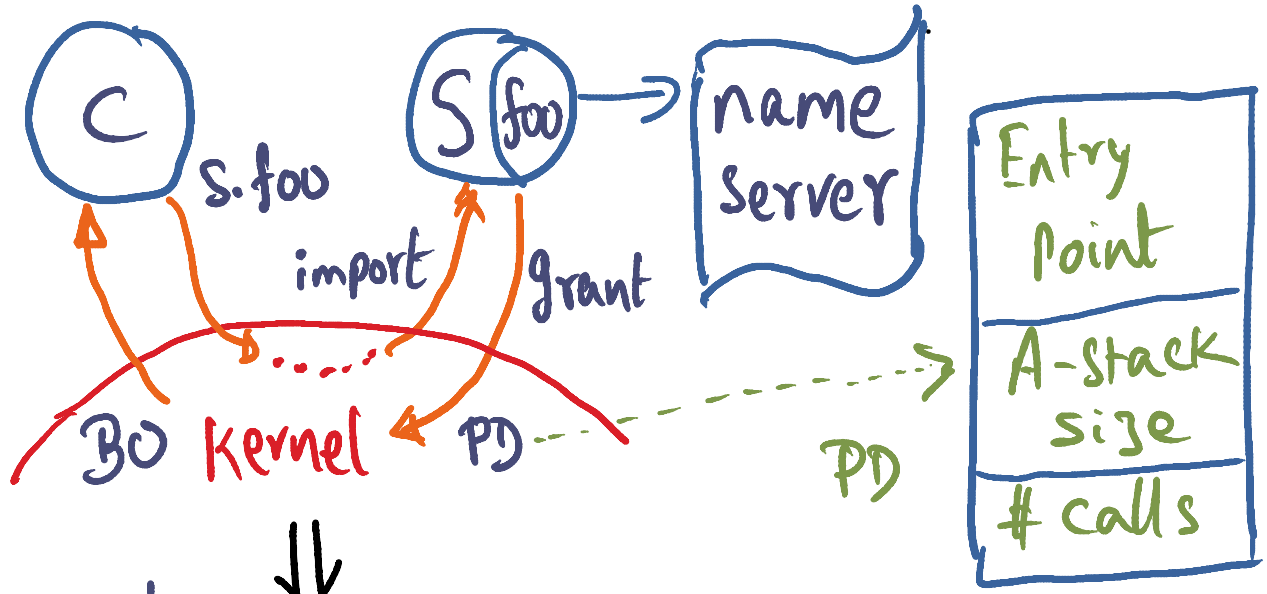
- Set up (Binding) ⇒ one time cost

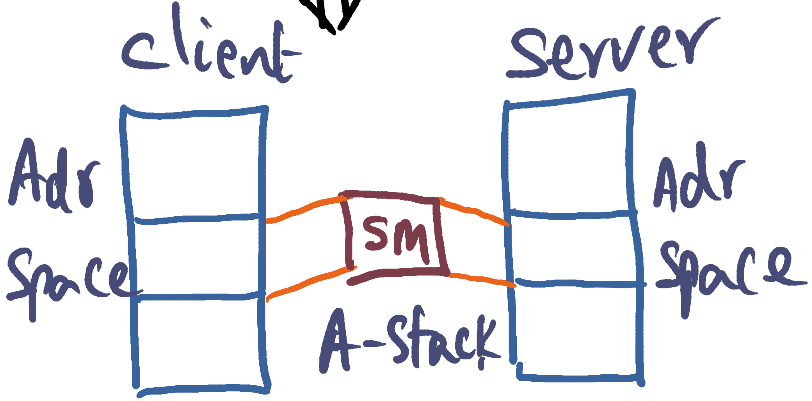


Making RPC cheap (Binding)

How to remove overheads?

- Set up (Binding) ⇒ one time cost

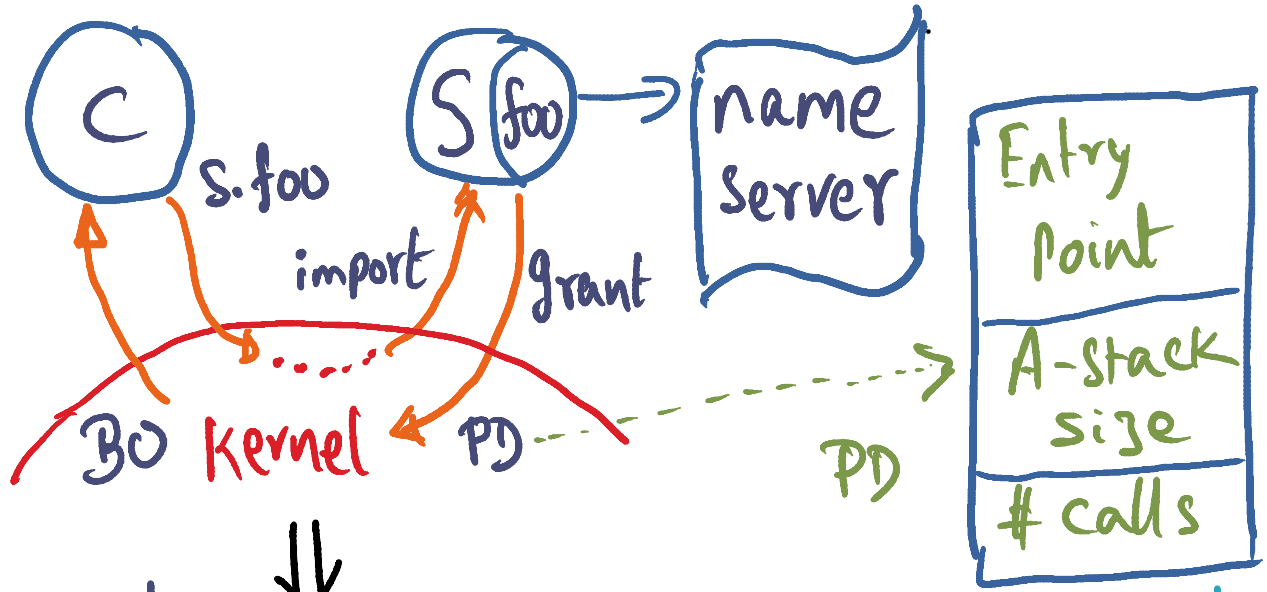


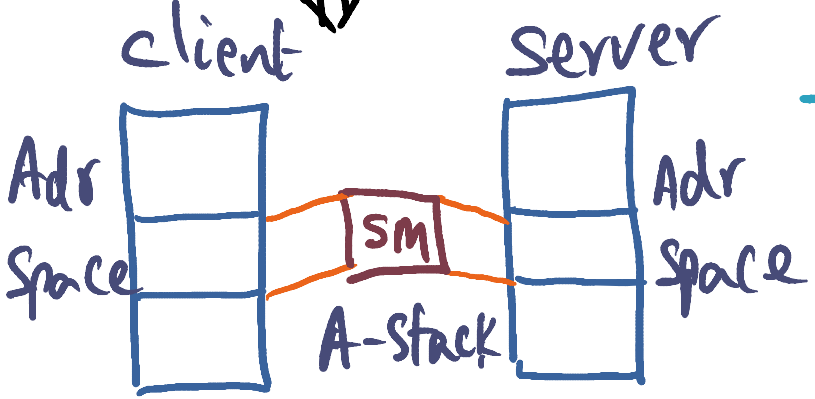


Making RPC cheap (Binding)

How to remove overheads?

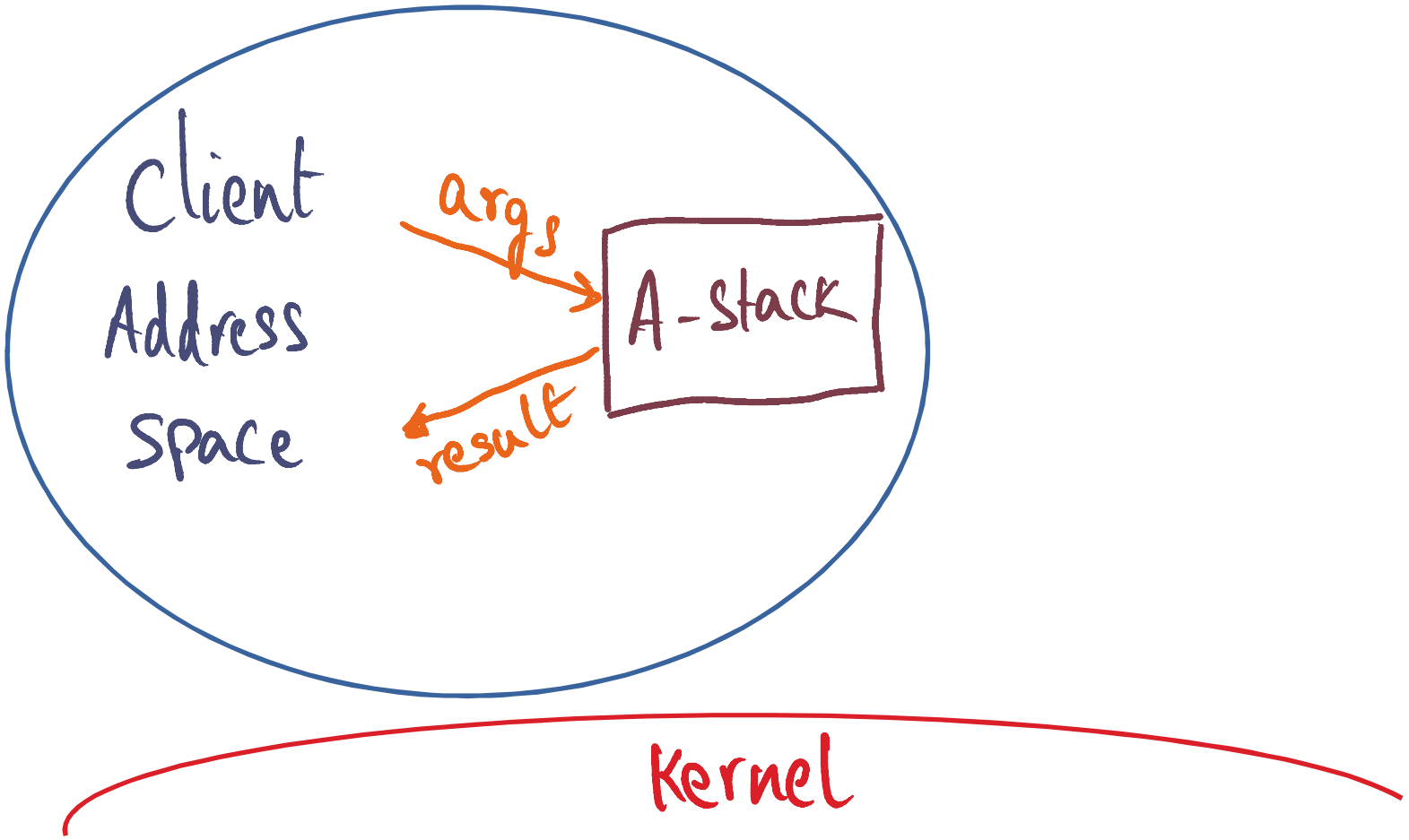
- Set up (Binding) ⇒ one time cost



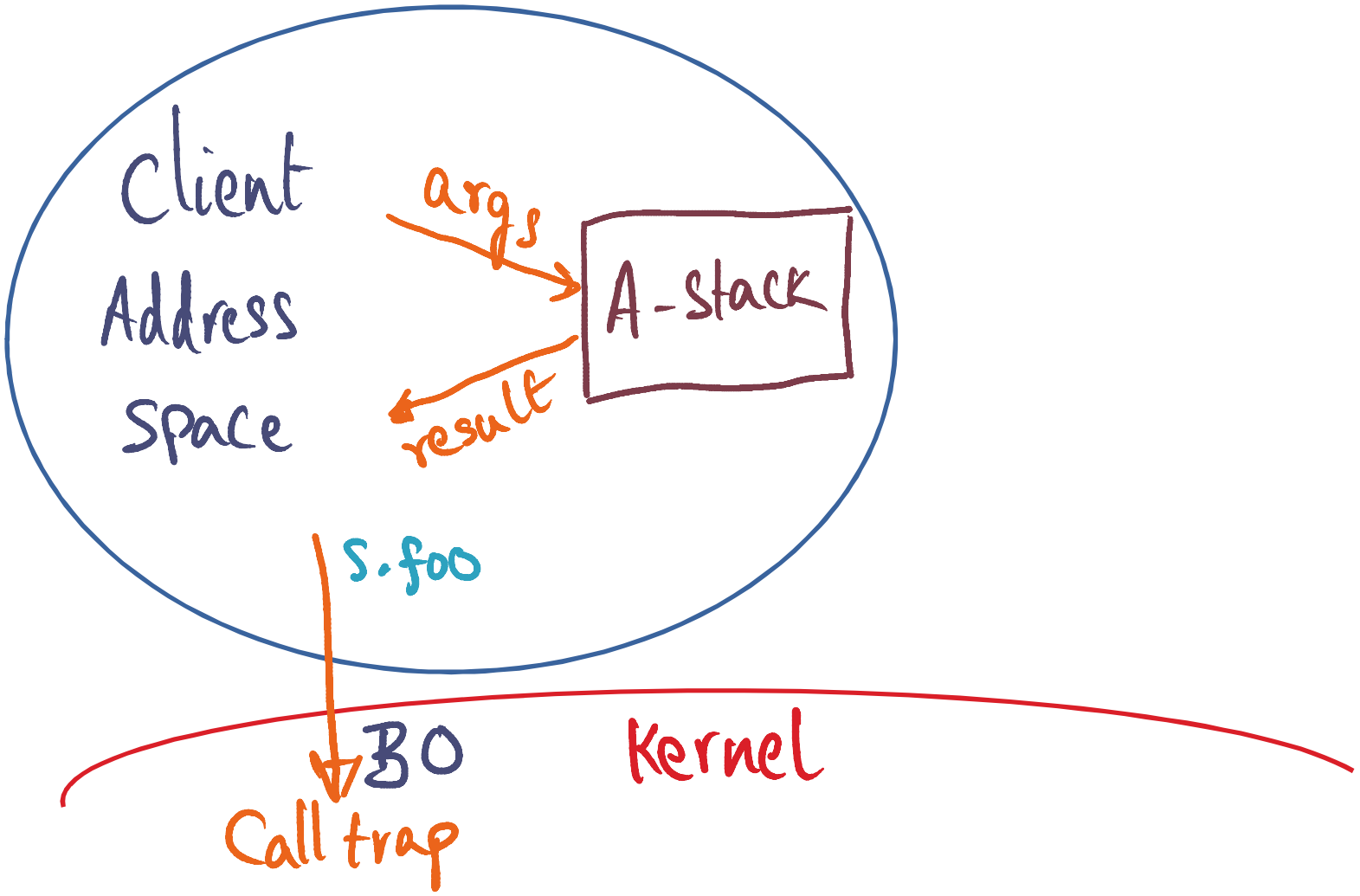


- Kernel mediation
  - * entry point set up
  - * import from client

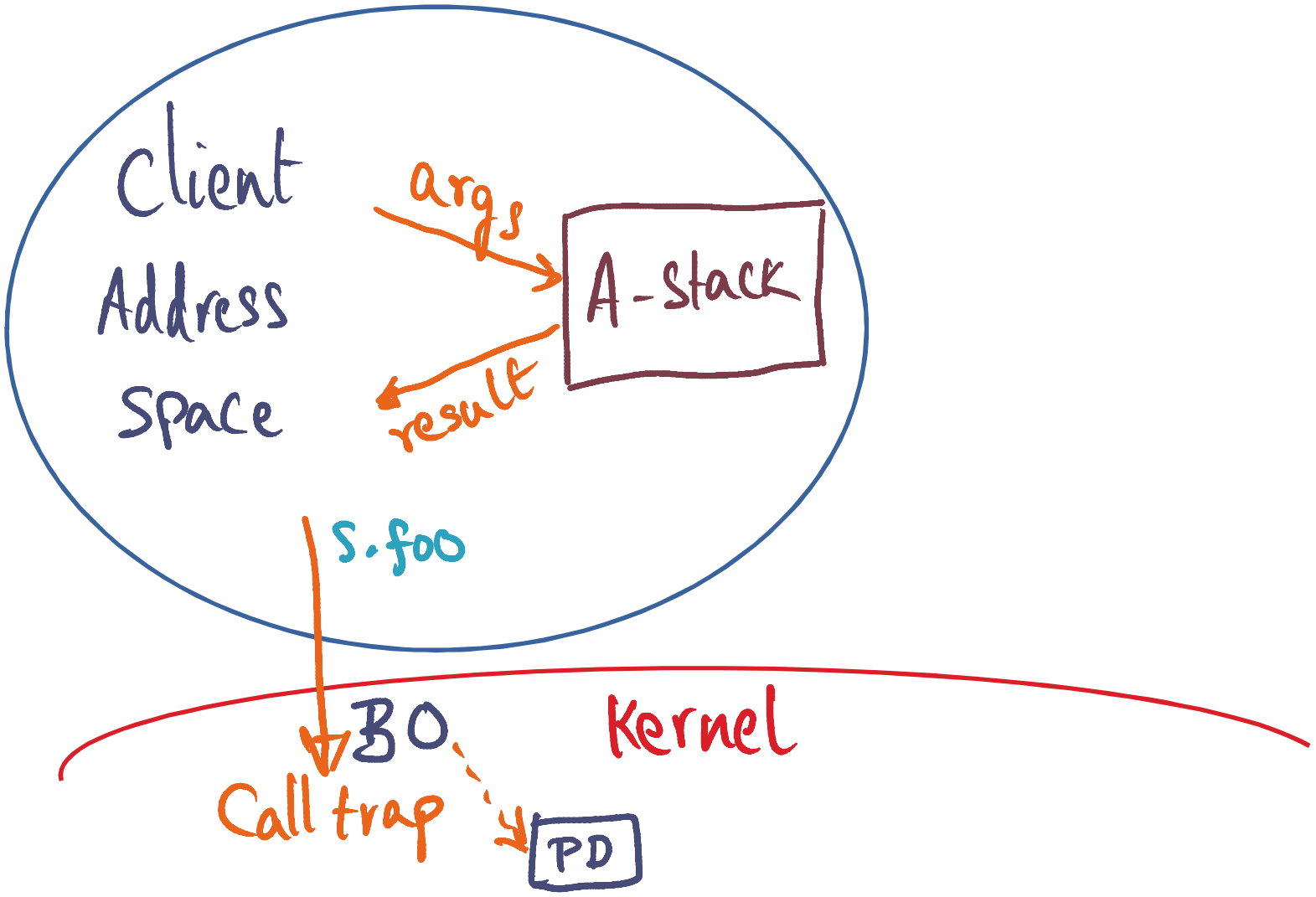
Making RPC cheap (Actual Calls)



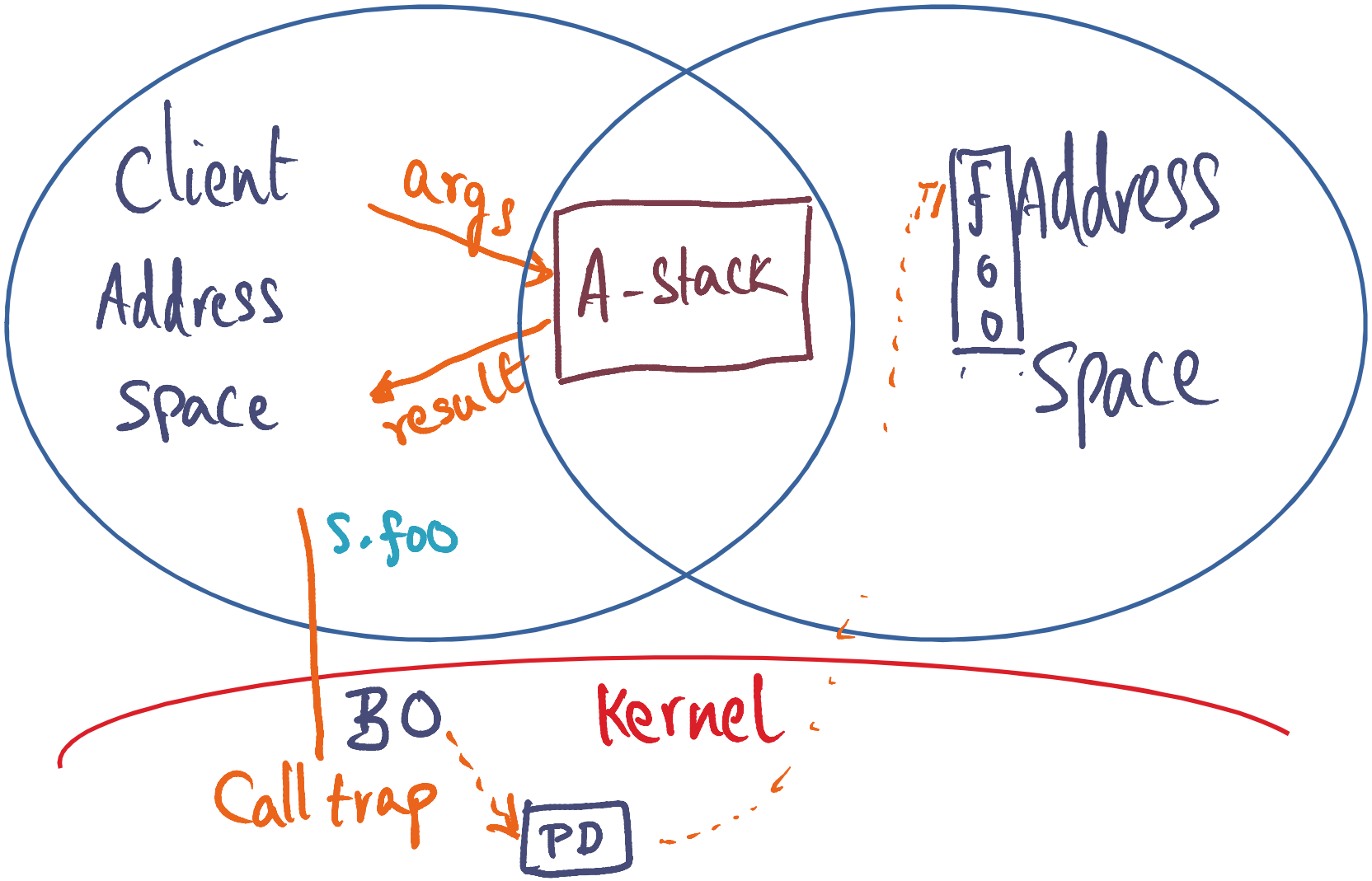
Making RPC cheap (Actual Calls)



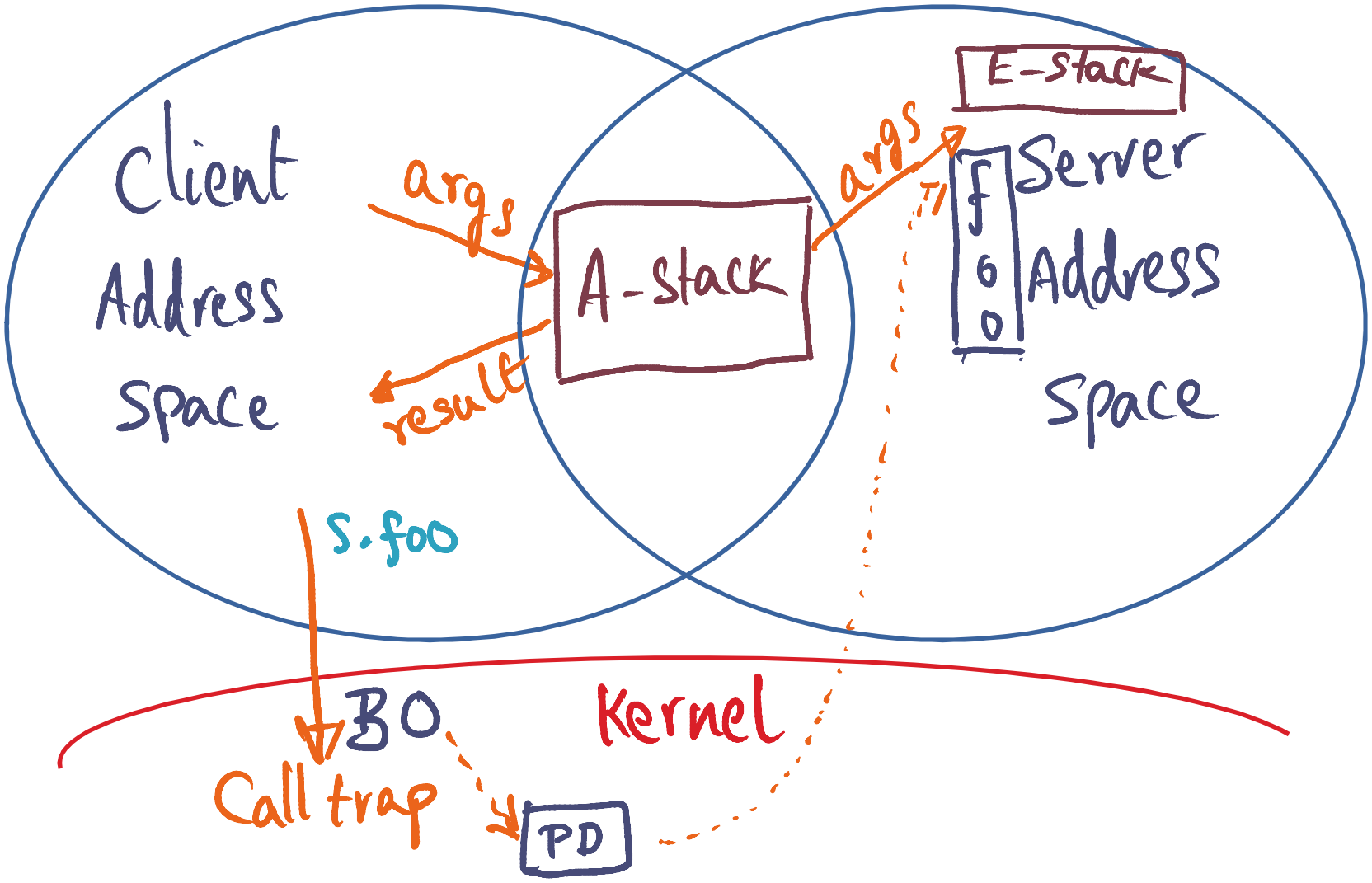
Making RPC cheap (Actual Calls)



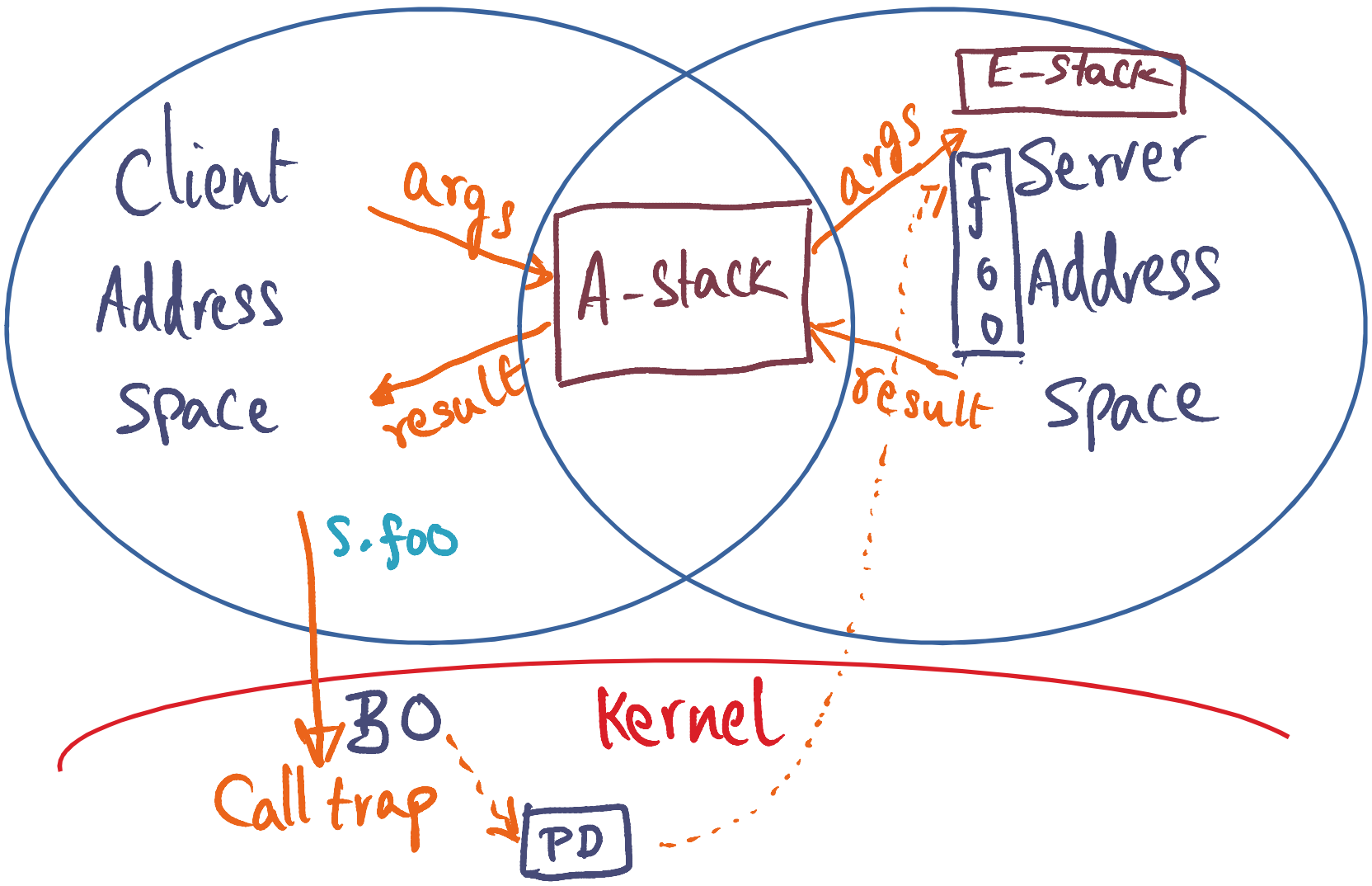
Making RPC cheap (Actual Calls)



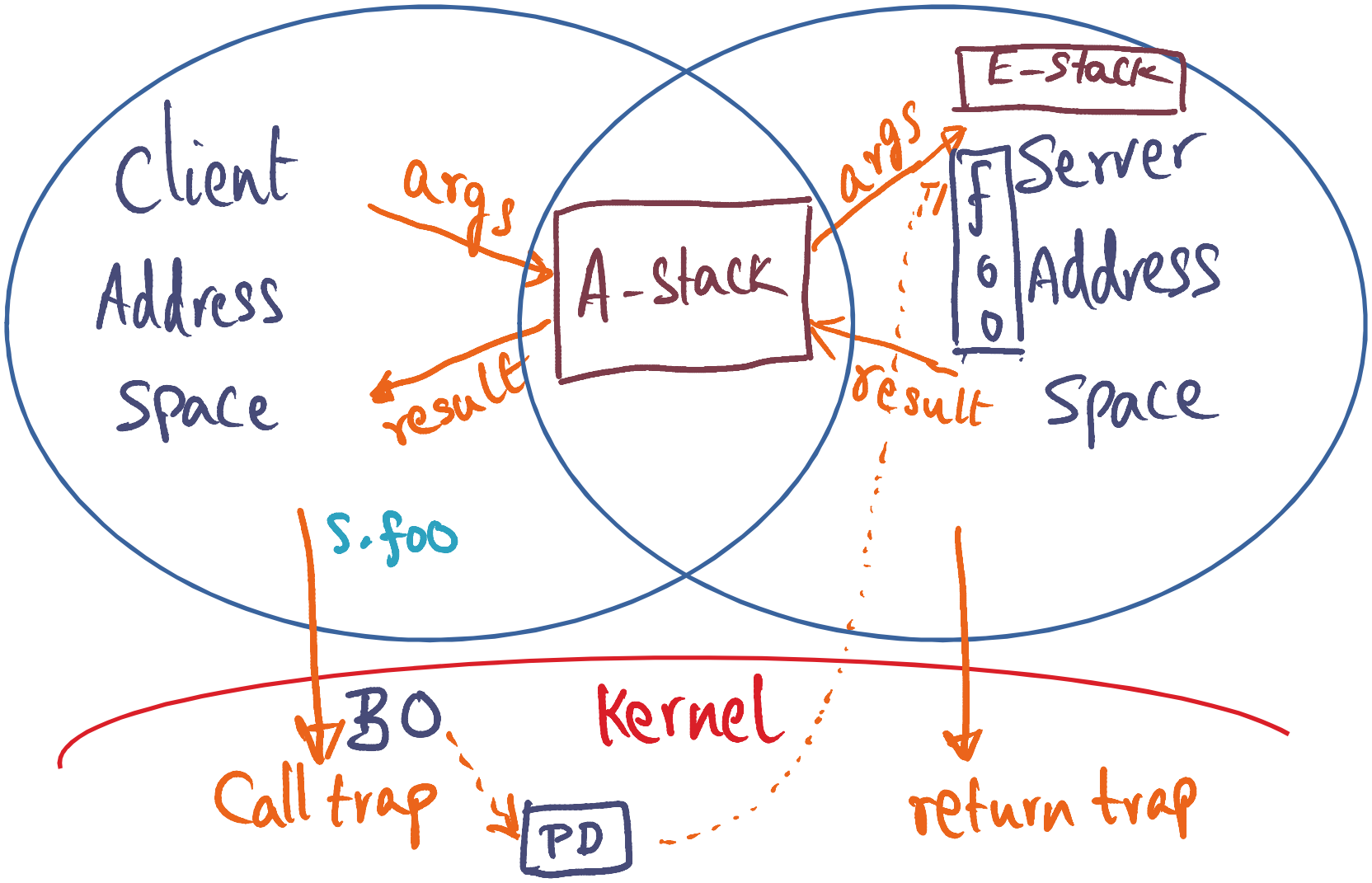
Making RPC cheap (Actual Calls)



Making RPC cheap (Actual Calls)



Making RPC cheap (Actual Calls)



Making RPC cheap (Actual Calls)

Original

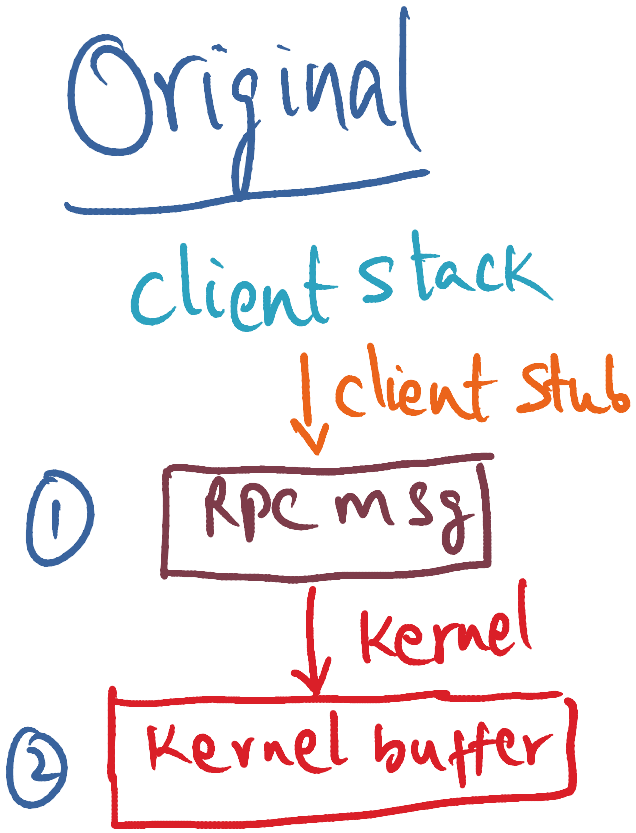
client stack

↓ client stub

① RPC msg

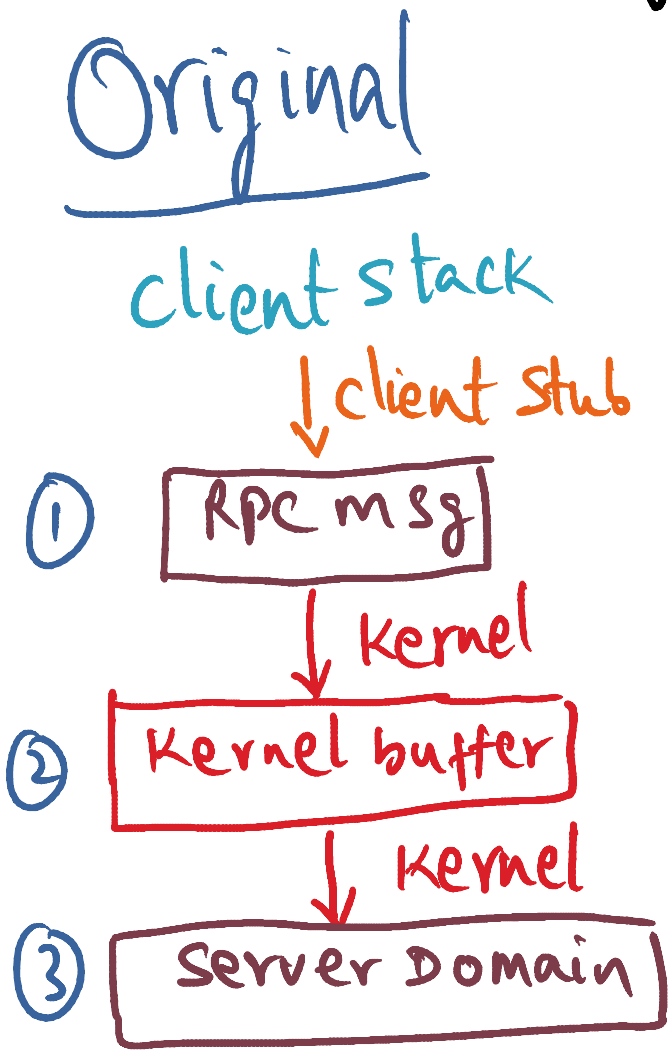
Making RPC cheap (Actual Calls)

Original

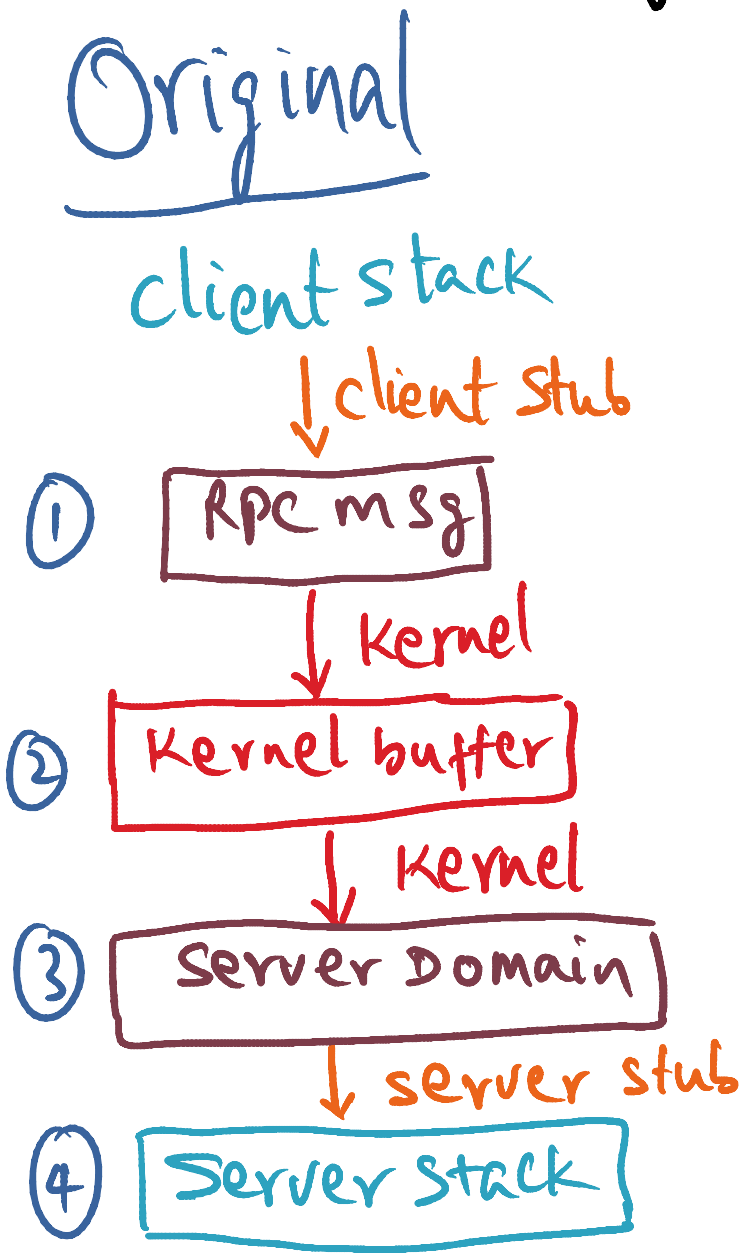


Making RPC cheap (Actual Calls)

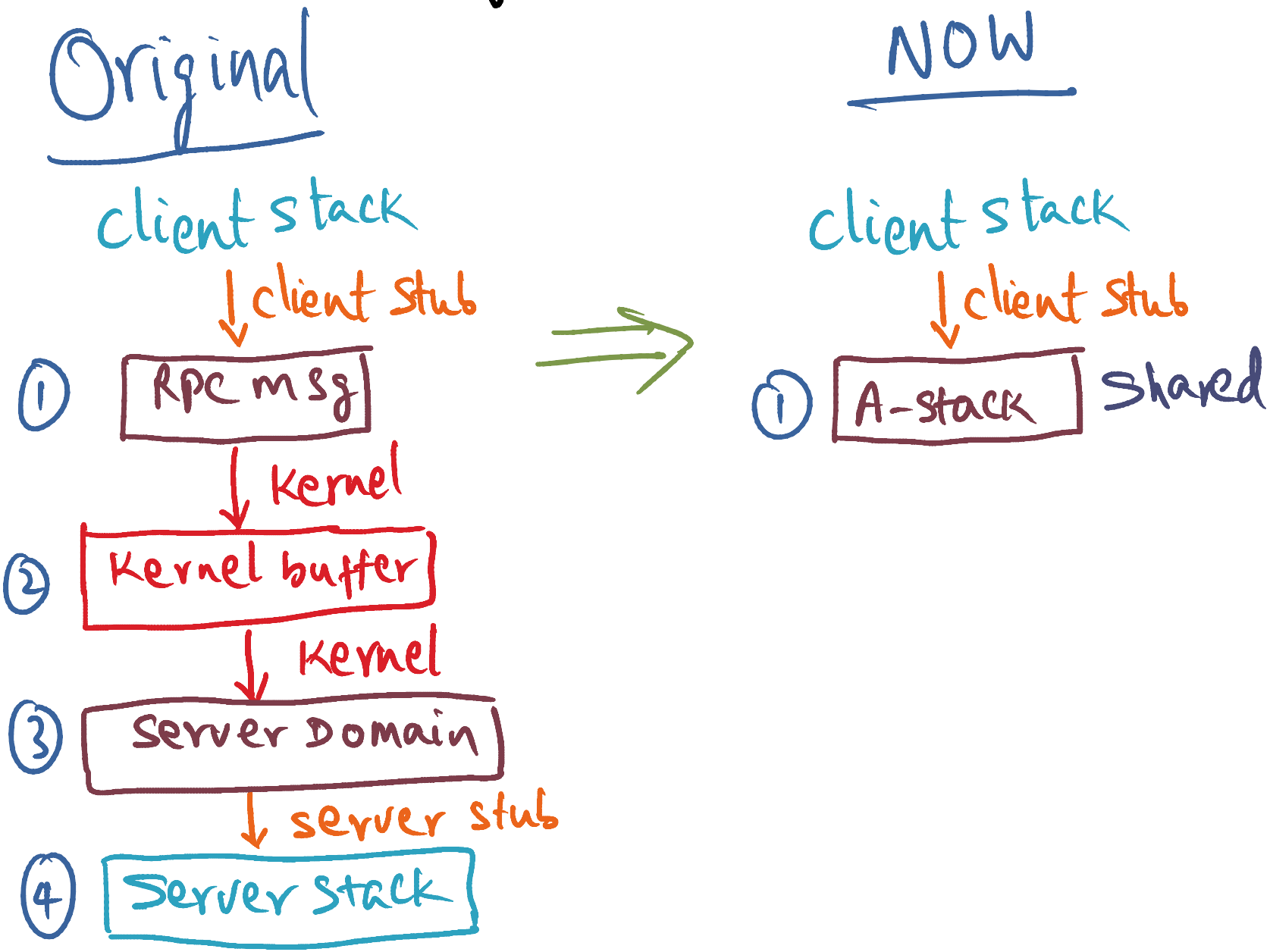
Original



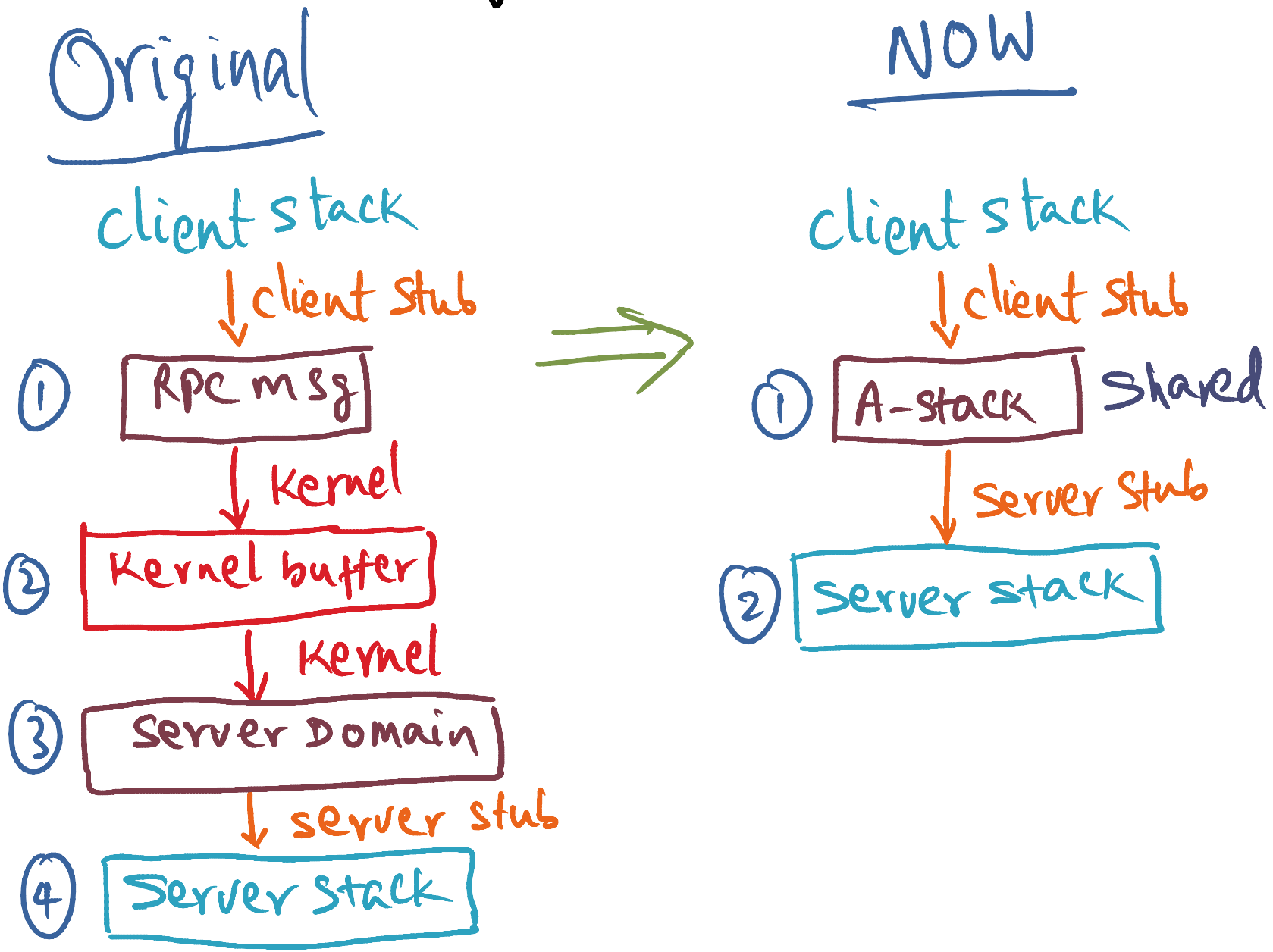
Making RPC cheap (Actual Calls)



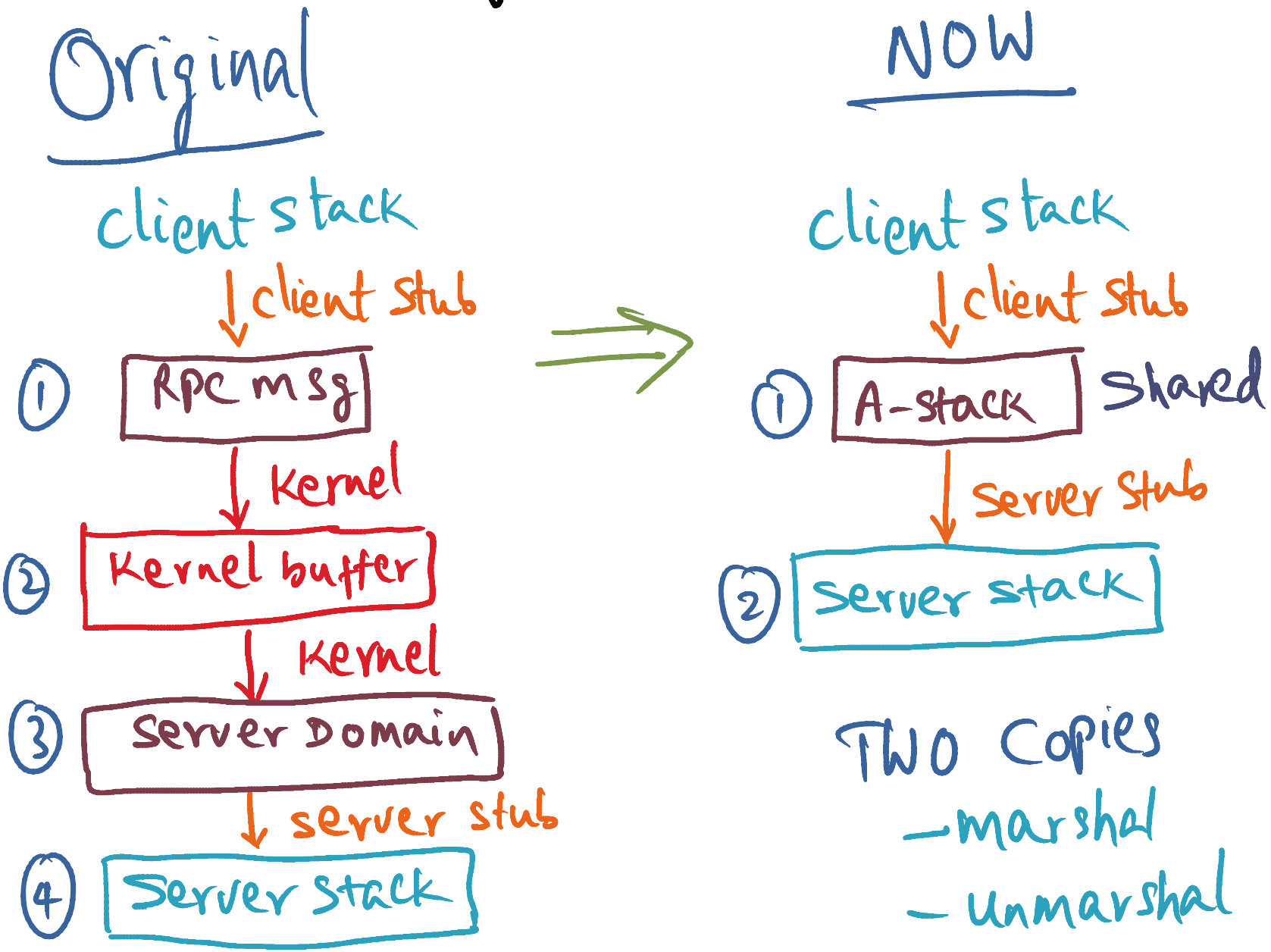
Making RPC cheap (Actual Calls)



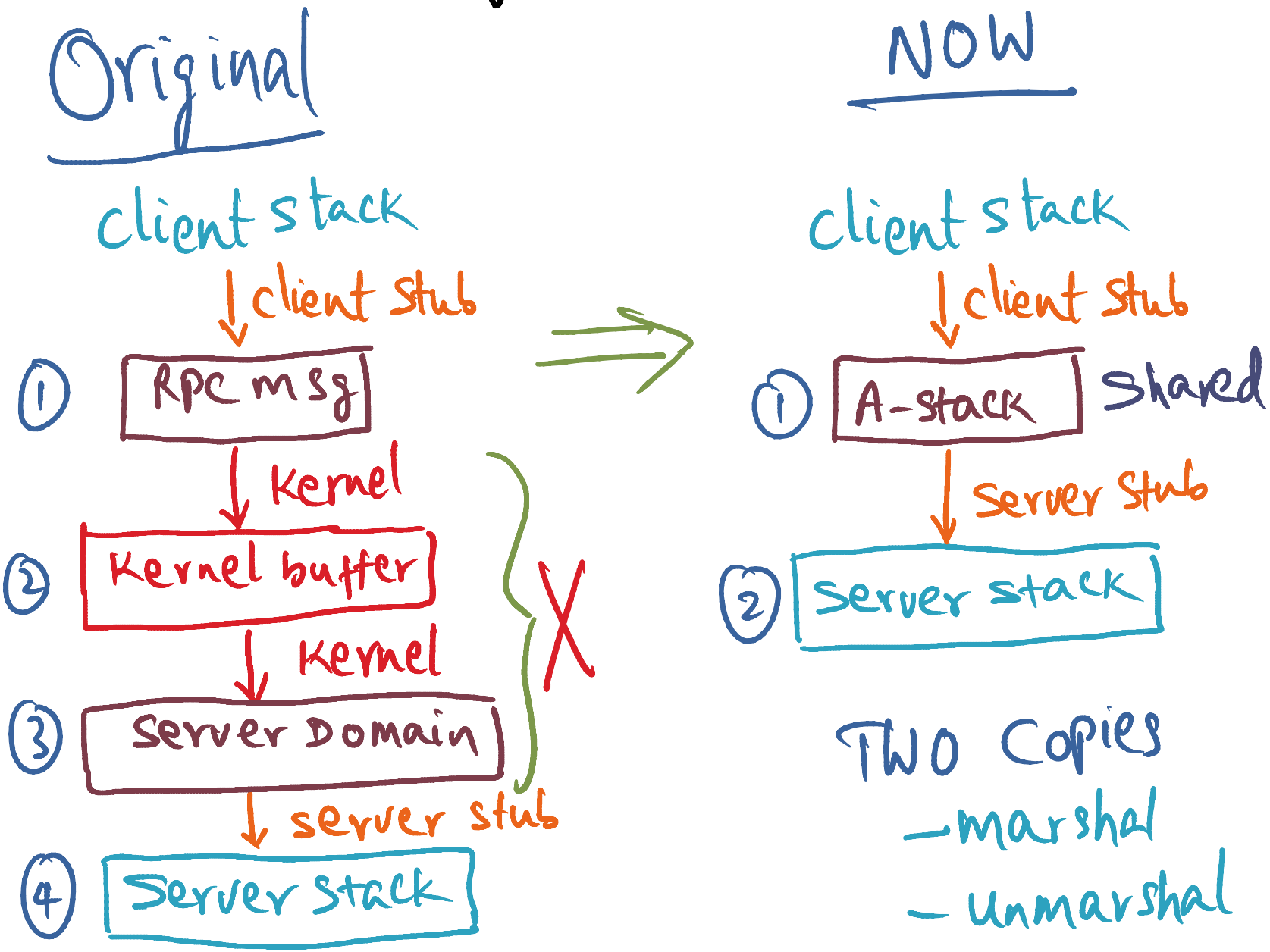
Making RPC cheap (Actual Calls)



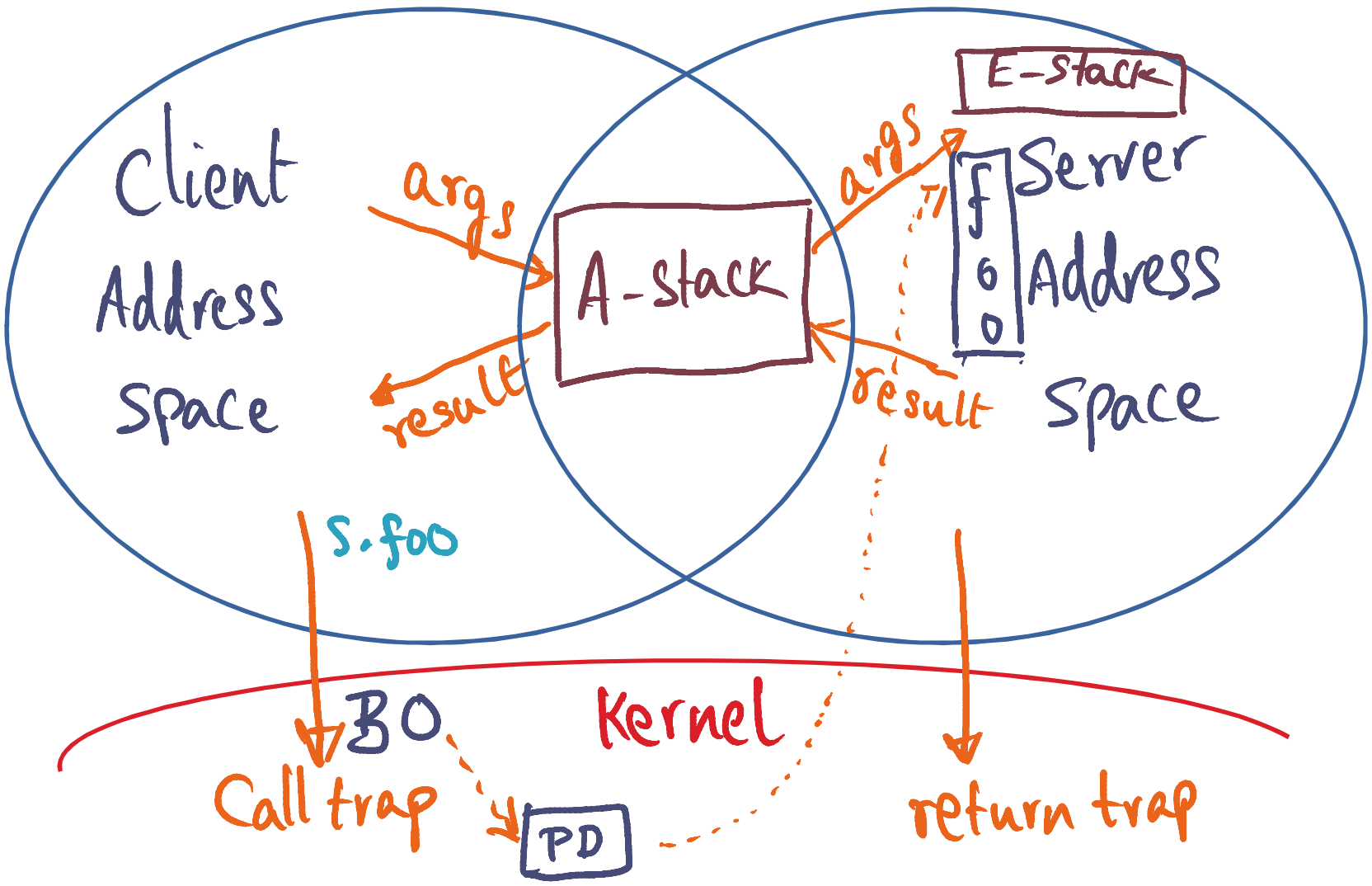
Making RPC cheap (Actual Calls)



Making RPC cheap (Actual Calls)



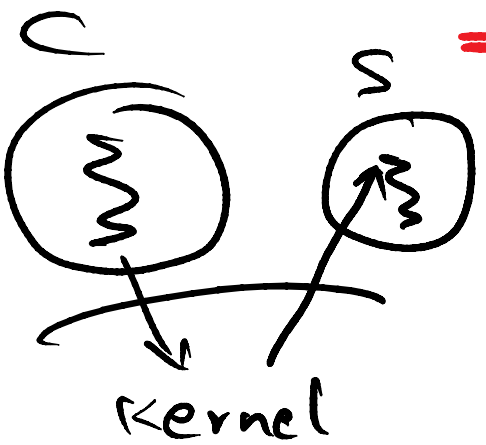
Making RPC cheap (Actual Calls)



Optimizations

Thread switching from client to server

* Doctor client thread to run in server domain



⇒ Save some explicit cost in thread switching

Implicit cost ??

⇒ use Liedtke's domain packing tricks

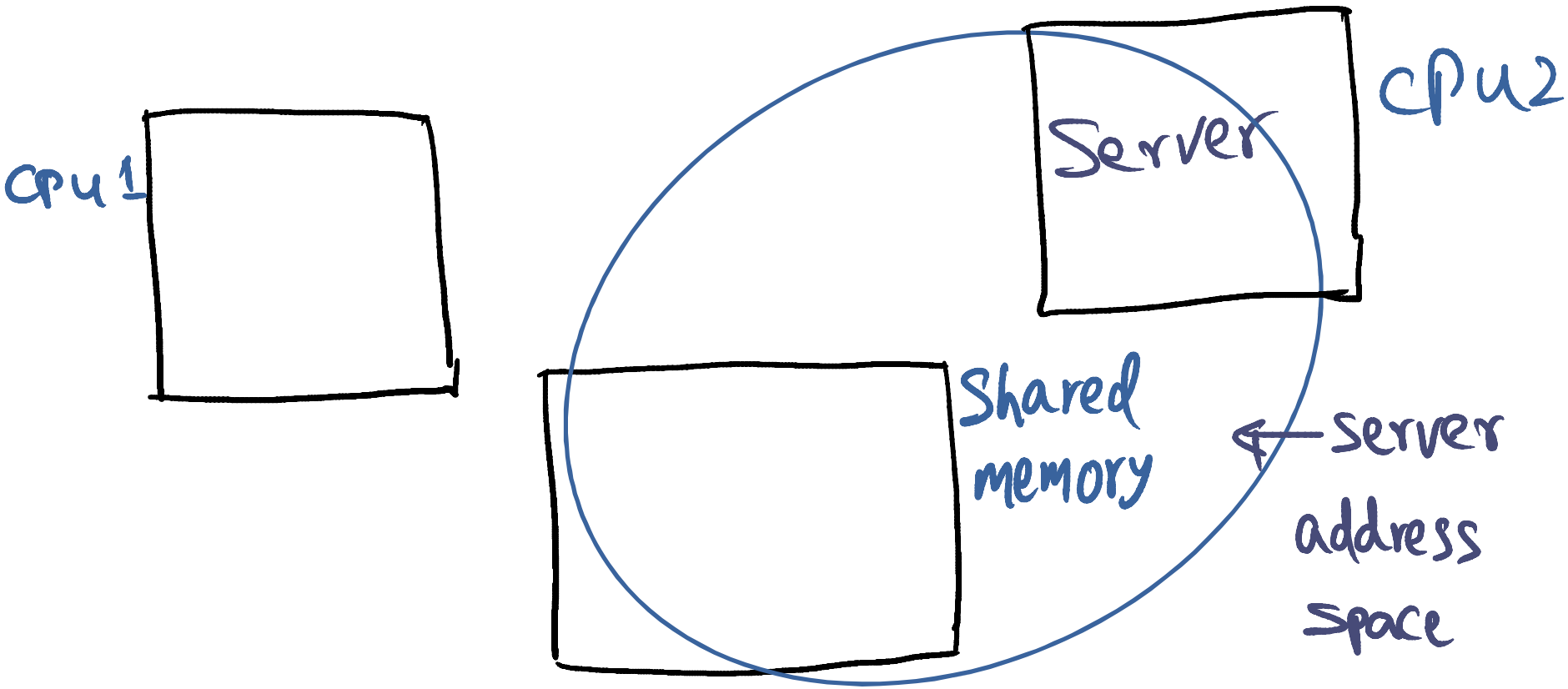
Summary

- One time binding cost
- Actual call eliminate kernel copies
- We are left with
  - Client/server traps (unavoidable)
  - Switching domains (optimizations to reduce explicit costs)
  - Loss of locality due to domain switching (small domains use Liedtke's tricks to reduce implicit costs)

RPC on SMP

Exploit multiple processors

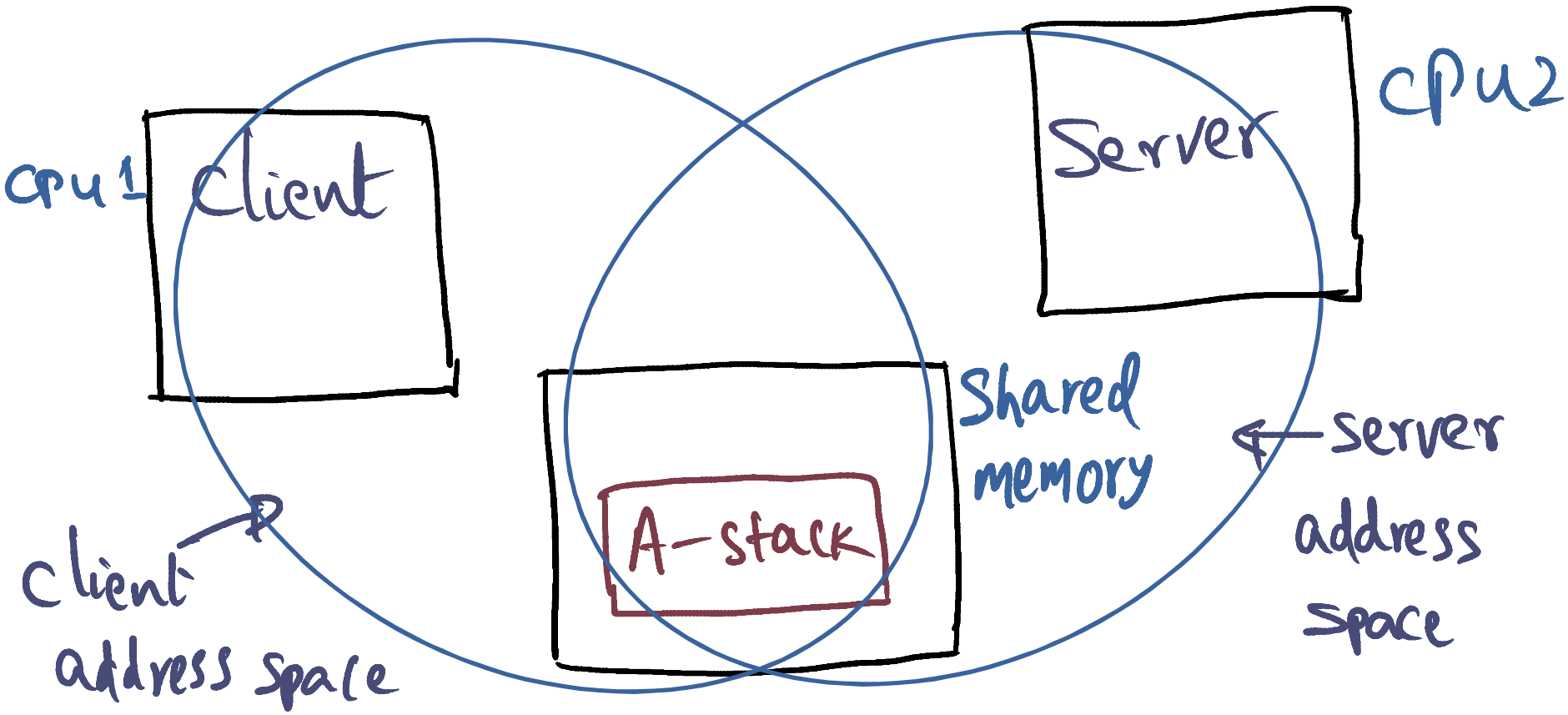
* Pre-load server Domain
* Keep caches warm



RPC on SMP

Exploit multiple processors

* Pre-load server Domain
* Keep caches warm



RPC on SMP

Exploit multiple processors

* Pre-load server Domain
* Keep caches warm