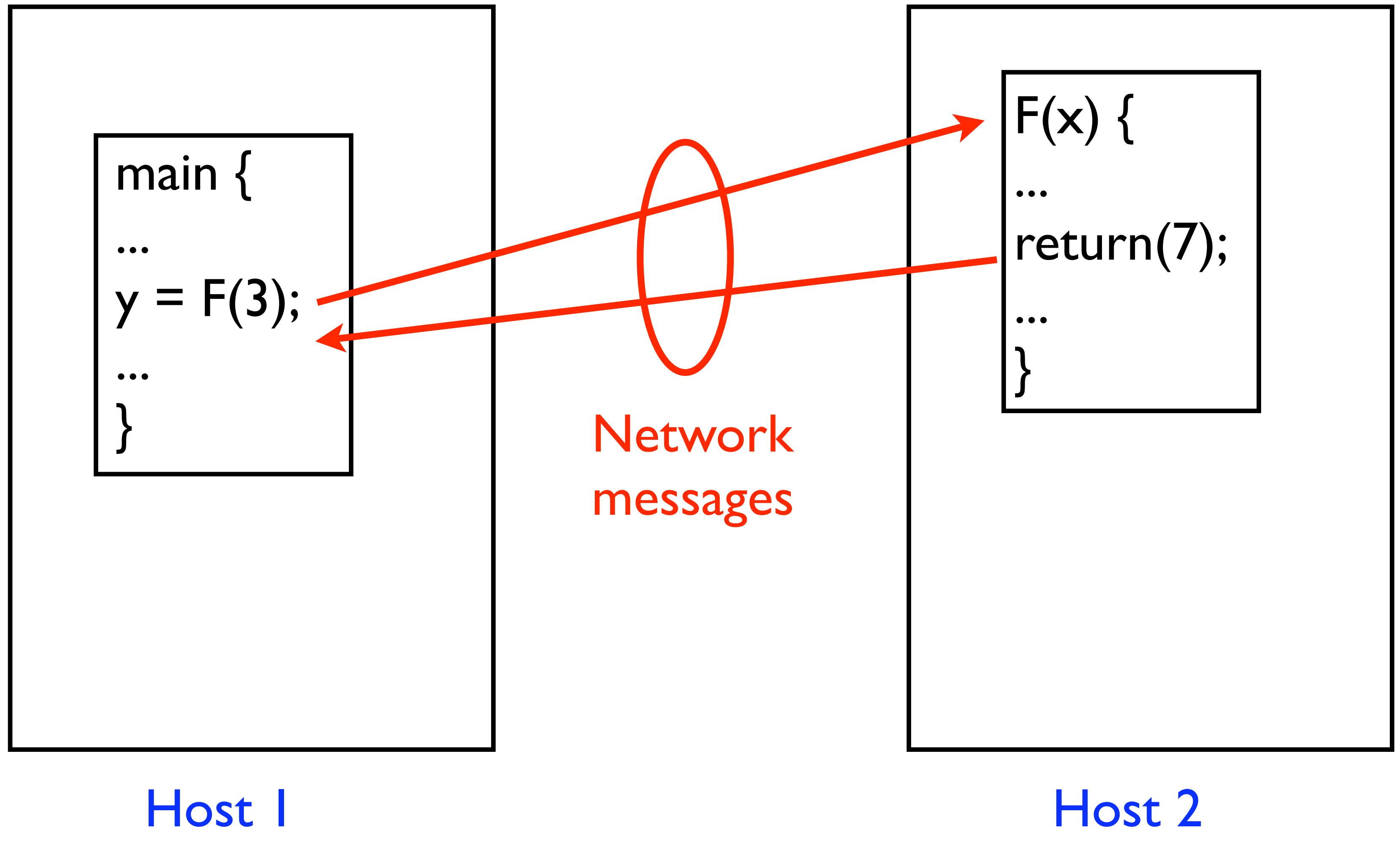
Scaling Up ...

- Remote Procedure Call
  - distributed applications, B2B, ...
- Distributed Objects
- Message-Oriented Middleware (MOM)
  - workflow, relaxed transaction models, ...
- J2EE and EJBs
- Web Services protocols

These Slides

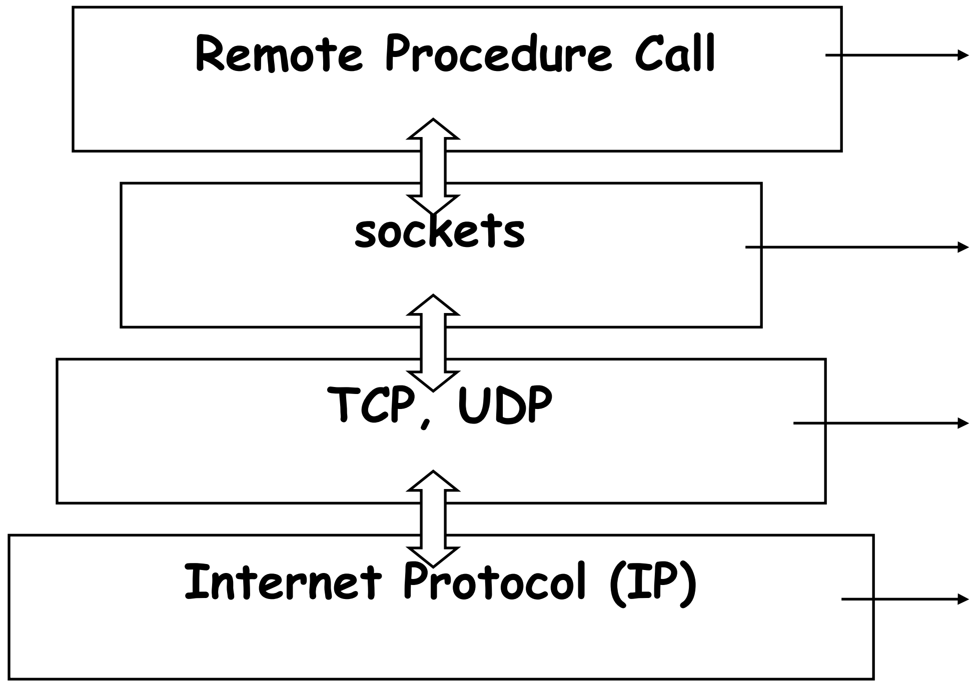
- RPC Systems
  - [BN97] Ch 3
  - [ACKM04] Sec 2.2
- TP Monitors
  - [BN97] Ch 2,
  - [ACKM04] Sec 2.3
  - [BN97] Ch 9 (2PC)

Remote Procedure Call (RPC)



- Caller sends *invocation message*
- Blocks until receives *reply message*

RPC as Layer of Abstraction



<u>Remote Procedure Call</u>:
hides communication details behind a procedure call and helps bridge heterogeneous platforms

<u>sockets</u>:
operating system level interface to the underlying communication protocols

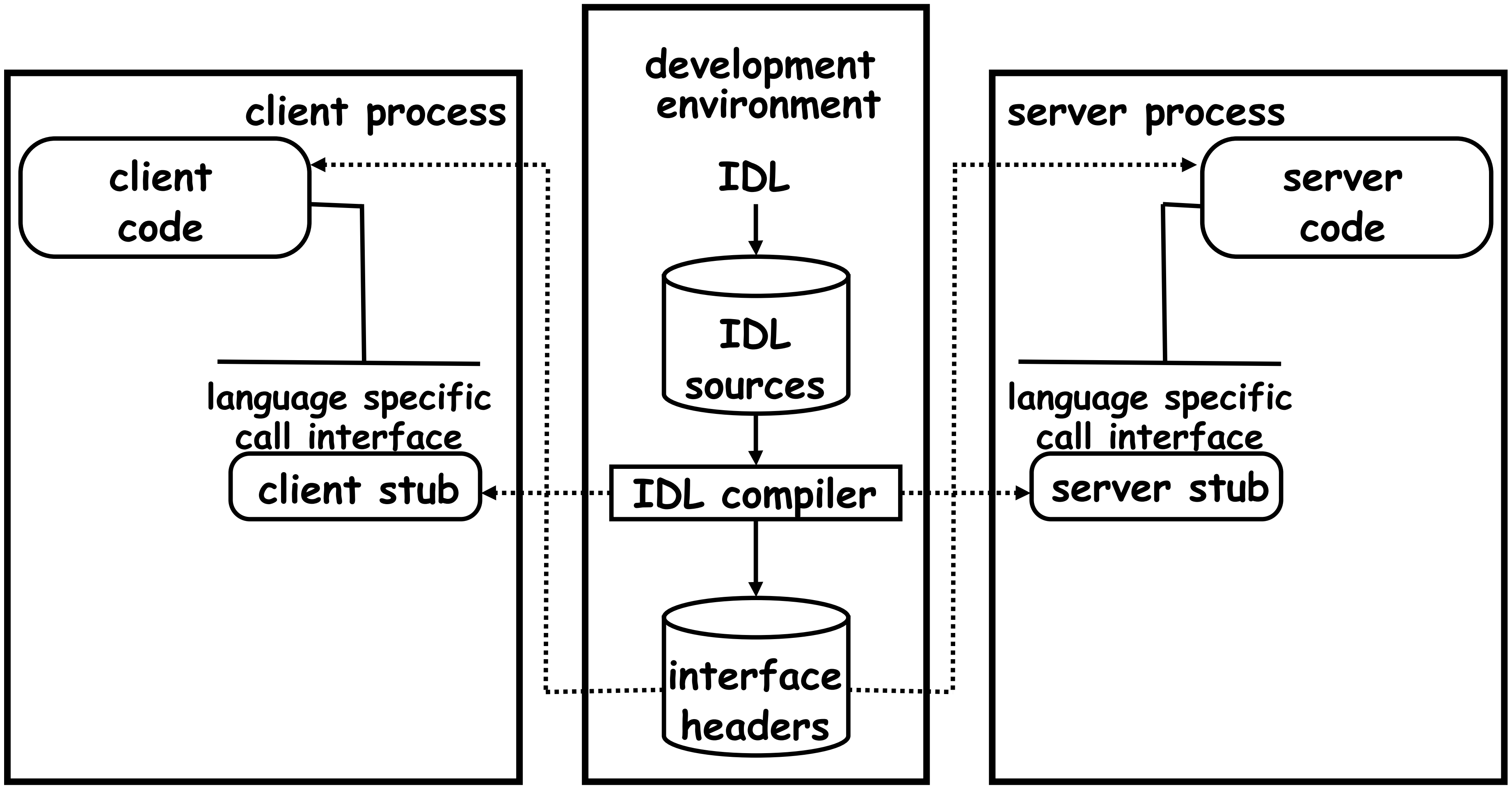
<u>TCP, UDP</u>:
User Datagram Protocol (UDP) transports data packets without guarantees
Transmission Control Protocol (TCP) verifies correct delivery of data streams

<u>Internet Protocol (IP)</u>:
moves a packet of data from one node to another

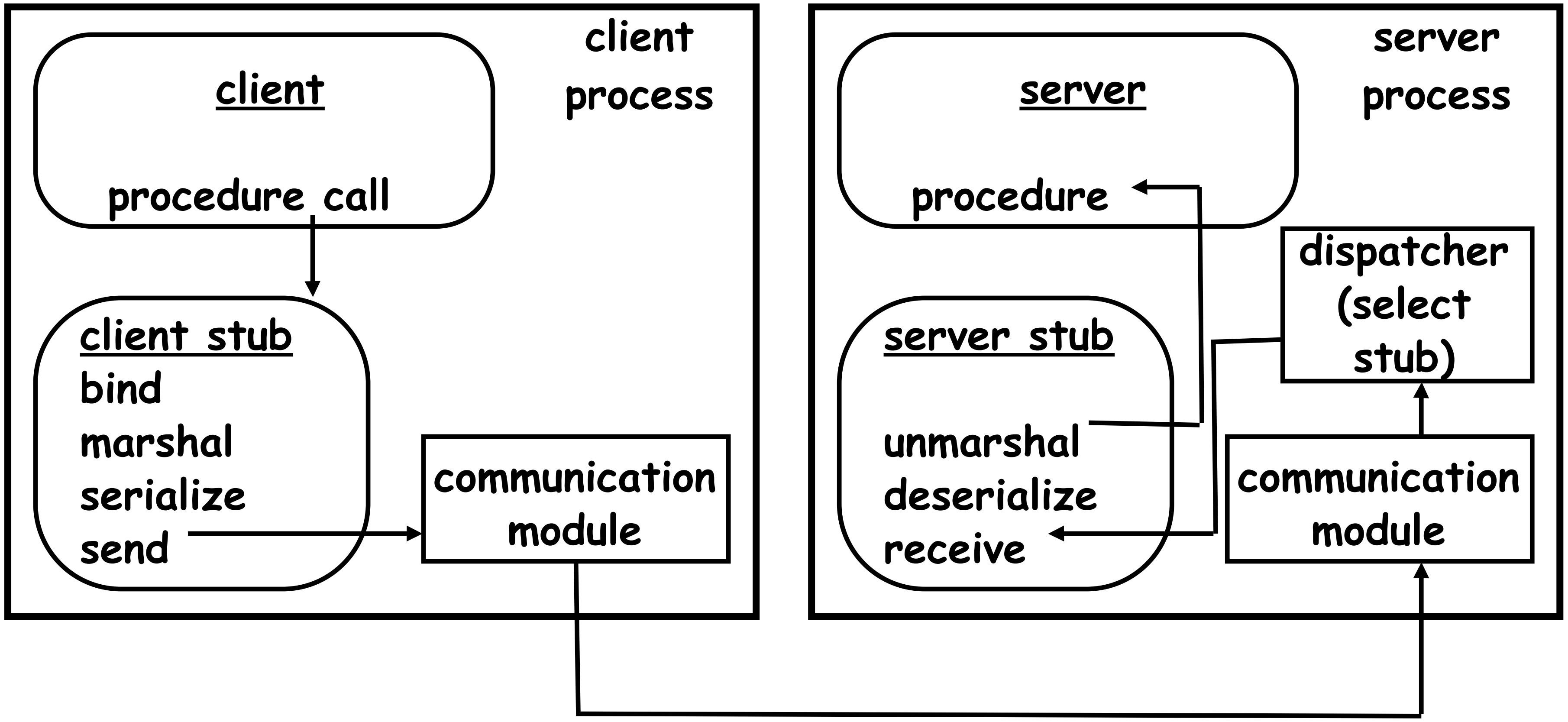
RPC Development

- Interface Description Language (IDL) in which signatures of procedures are described
- IDL Compiler generates caller & callee *stubs*
- Stubs are responsible for marshalling, transmitting and unmarshalling arguments and results

RPC Development



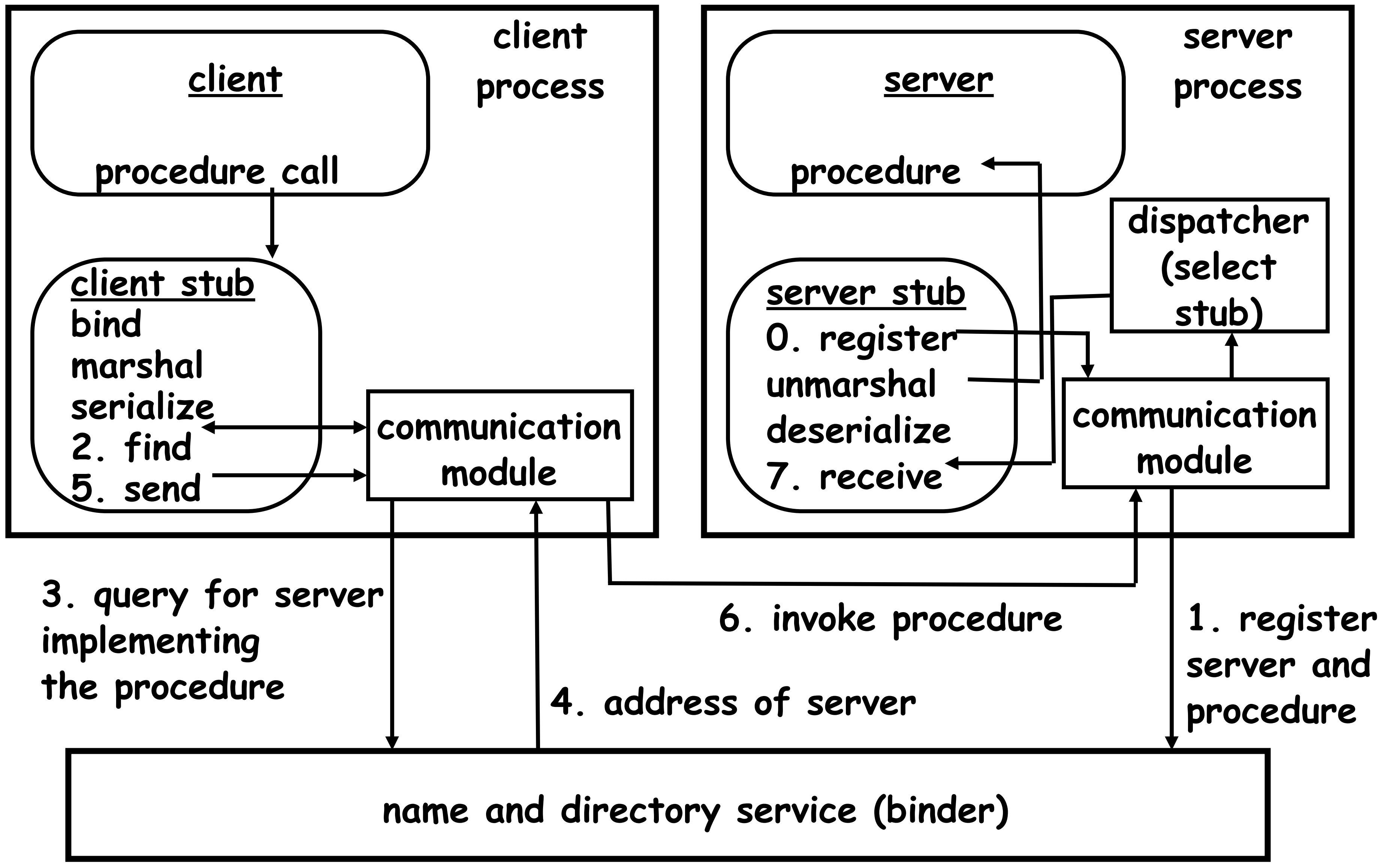
Basic RPC Runtime



Naming & Directory Service

- Callee registers with NDS
- Caller looks up callee by name & signature
- Possibly multiple matches
  - → *traders*
- Possibly multiple server instances
  - → potential for load balancing
- Possibly no active server instances
  - → start one?

Dynamic Binding for RPC



Parameter Translation

- Canonical encoding on wire
  - solves the "n**2 problem"
- Receiver-translates
  - best performance if homogeneous

Security

- Advantageous to build authentication into RPC infrastructure
- Discussion deferred until later

Fault Tolerance

- Retries
  - Reliable transport?
  - If idempotent $\longrightarrow$ want *at least once*
    - caller implements this
  - Not idempotent $\longrightarrow$ want *at most once*
    - server implements this
  - See *Transactional RPC* below

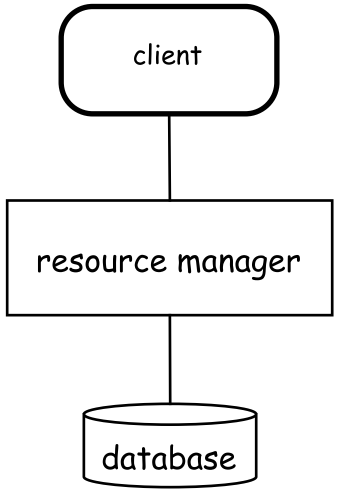
RPC Performance

- Procedure invocation overhead 100-1000 times greater for RPC than local call
- *Plus* the communication latency
- 15,000 machine instructions don't take very long these days ...
- But the speed of light is constant
    - and *slow* - 60ms RTTs are common

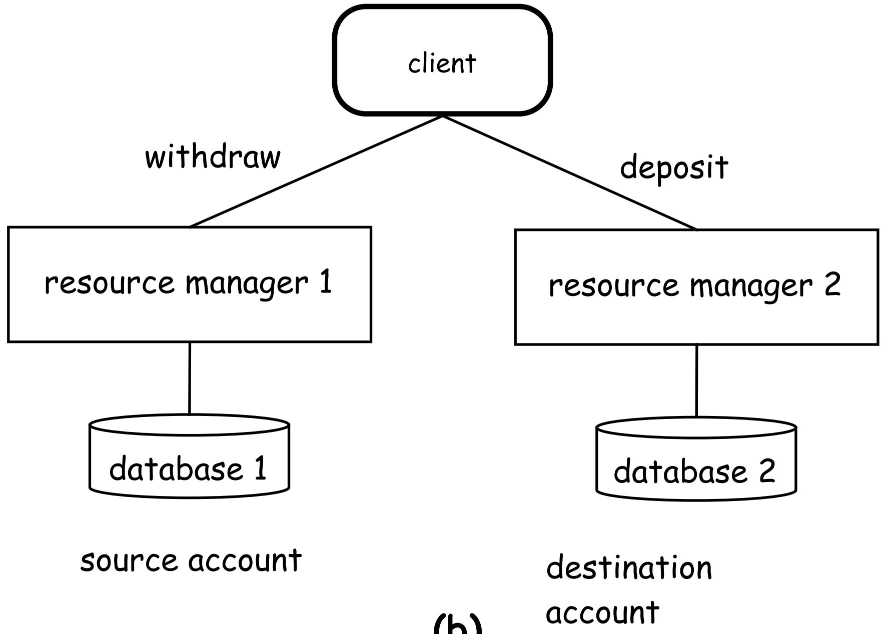
Transactional RPC

- Suppose RPCs done in application that requires transactional ACID properties
- Distribution makes this difficult ...

Distributed App Needs Distributed Commit



(a)



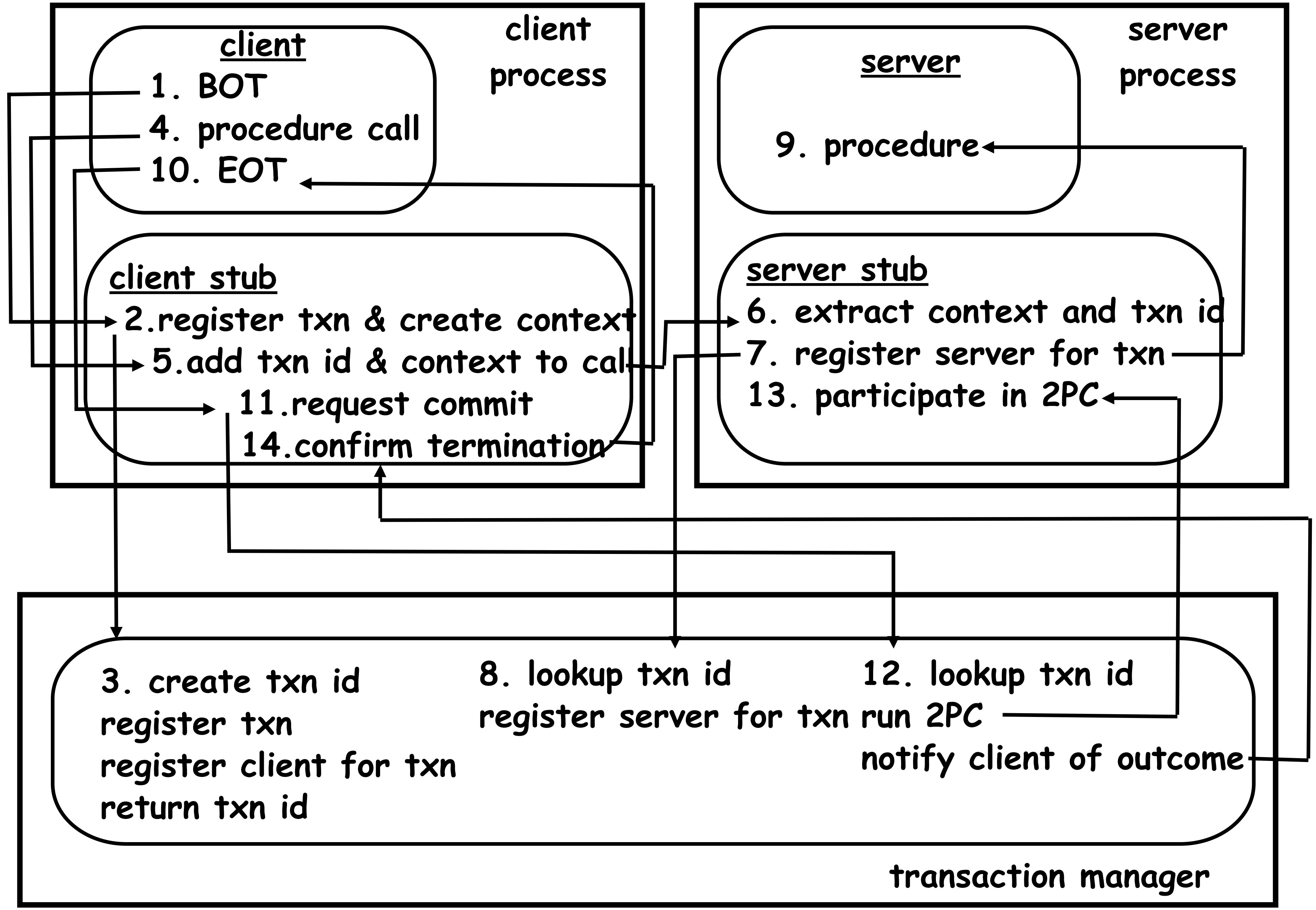
(b)

- RM1 and RM2 must *both* commit or *both* abort
- Failure model is not Byzantine or FailStop but *Crash-Recover*
- This looks like a job for 2PC!

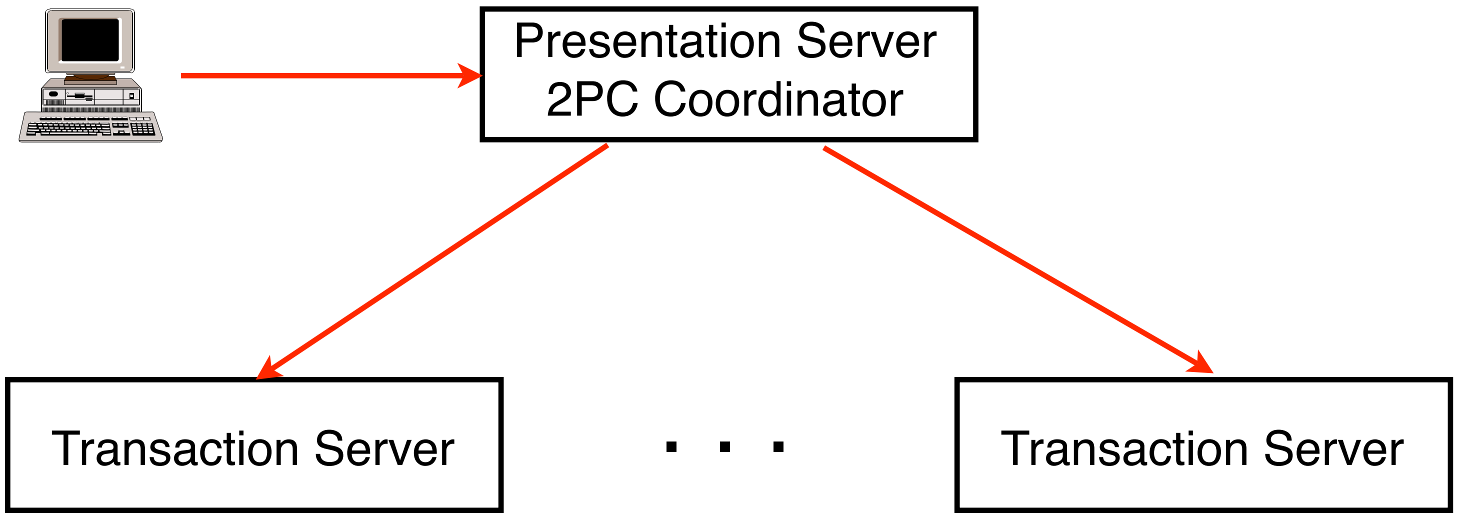
Early Solution: TP Monitor

- Transaction Manager
- Multiple RPCs between BOT-EOT calls execute as one (distributed) transaction
- Coordinator for 2-Phase Commit

Transactional RPC

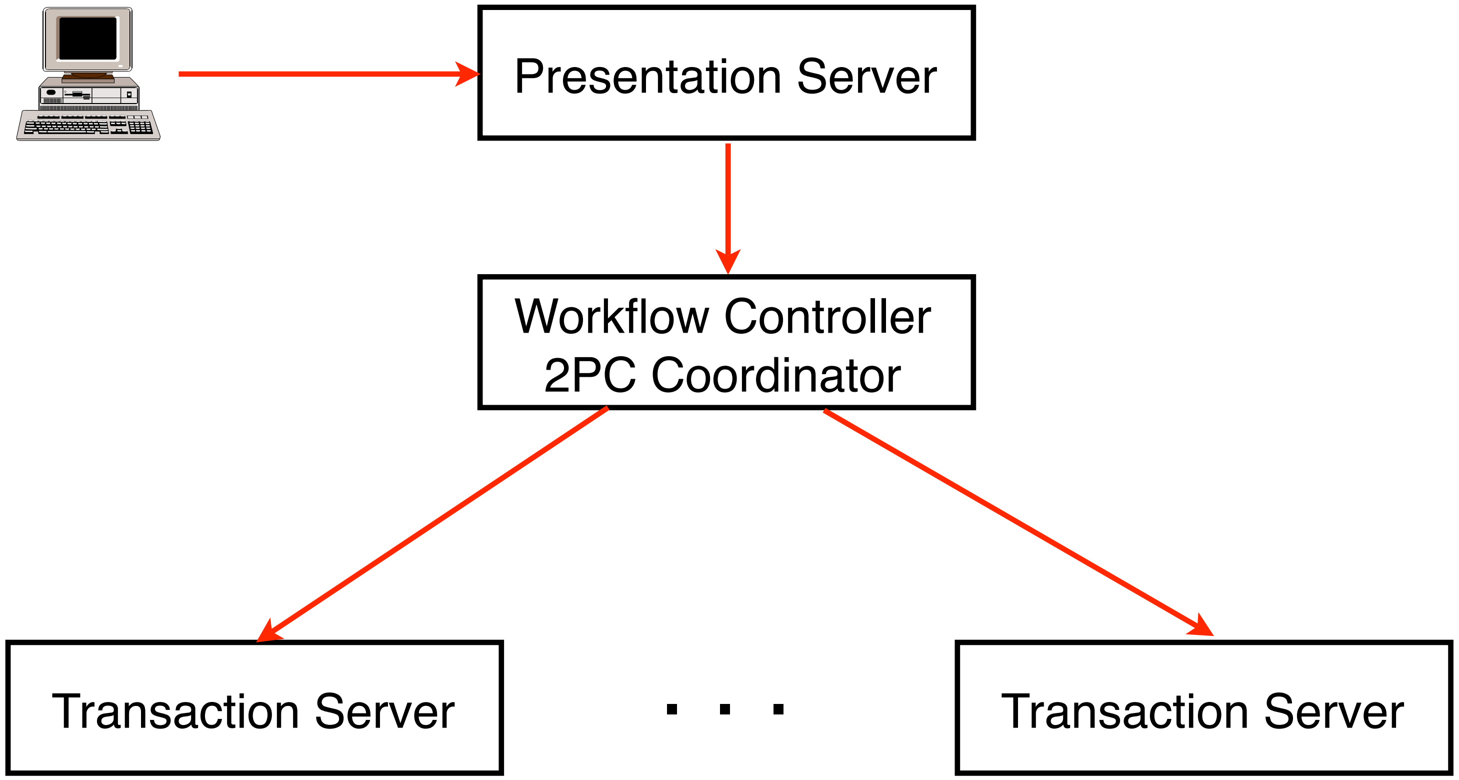


2-Tier TP Monitor Architecture



If there is more than one transaction server, the commit coordinator has to be in the upper tier ...

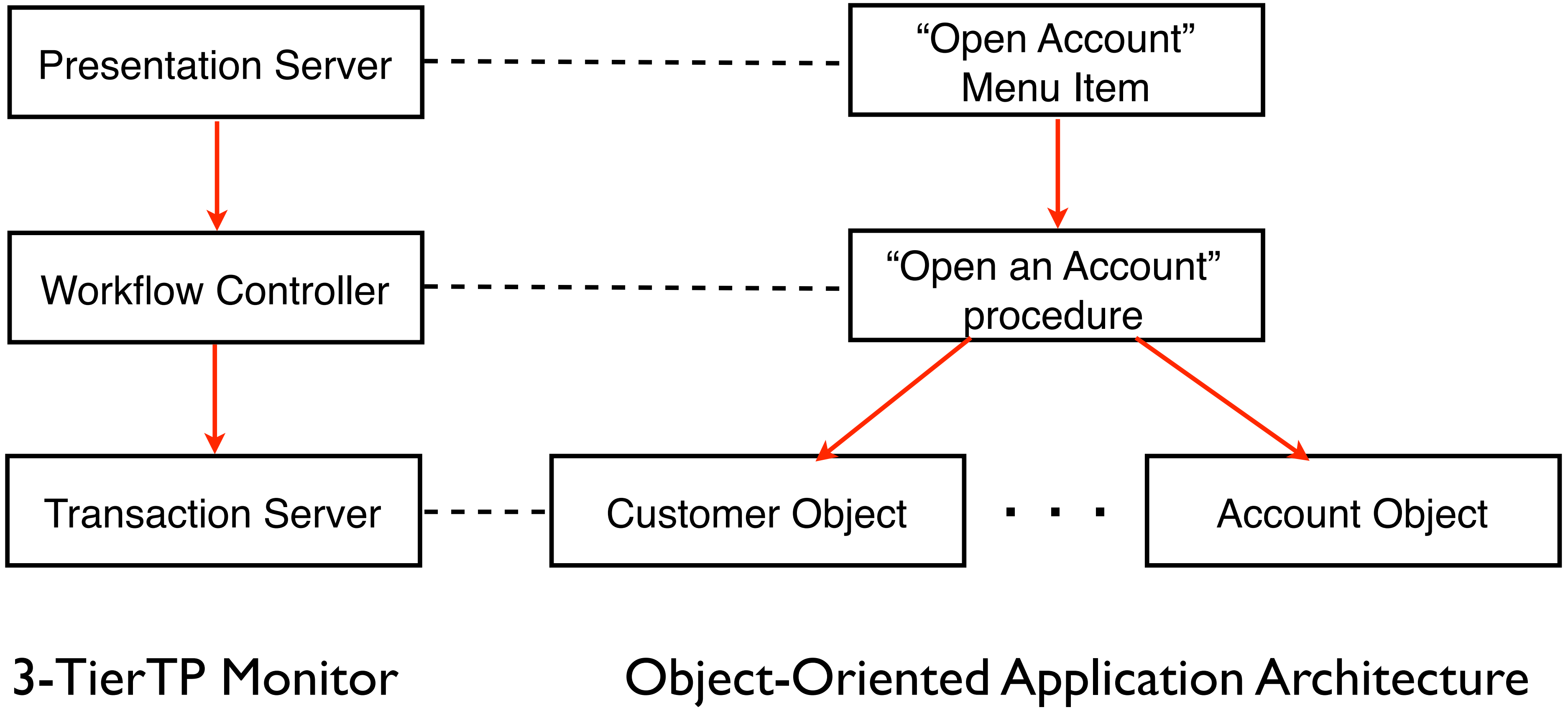
3-Tier TP Monitor Architecture



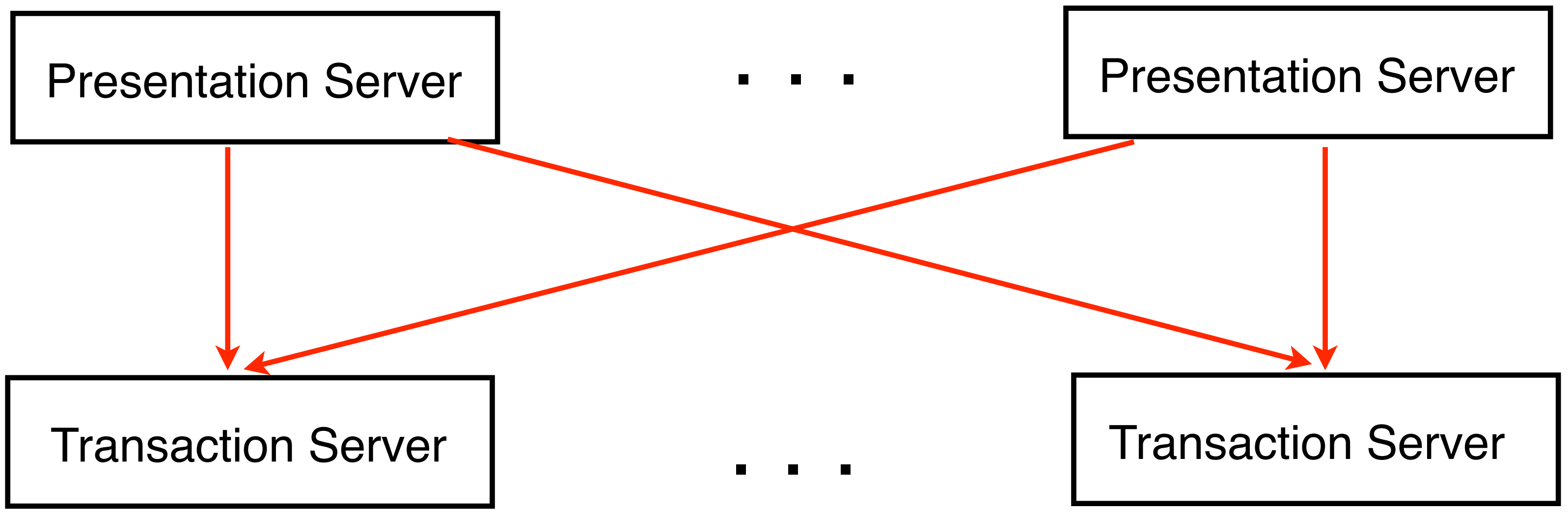
The 3 Tiers match the 3 application layers ...

3-Tier TP Monitor ...

- 3-Tier Model Maps to Object-Oriented Application ...

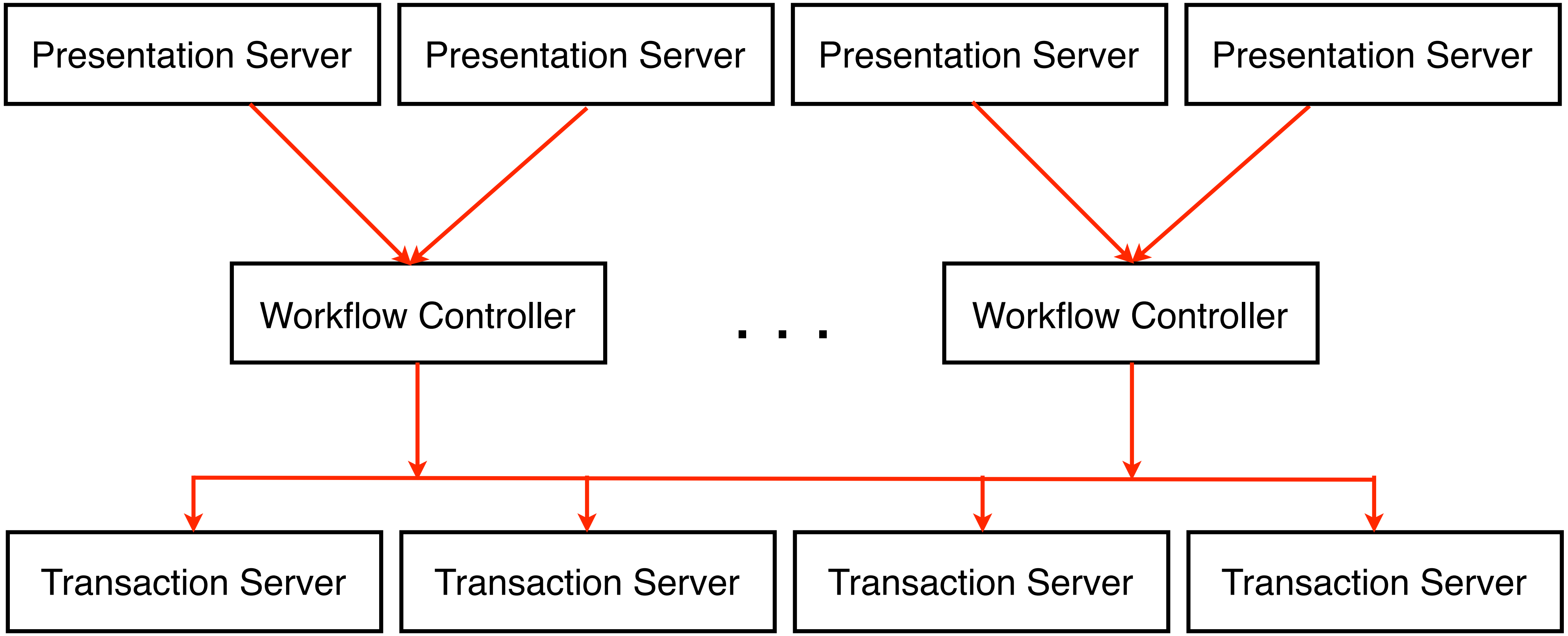


2-Tier Communication



2-Tier system requires quadratically many edges (sessions)

3-Tier Communication



3-Tier system requires only linearly many sessions