CLOUD COMPUTING CONCEPTS

with Indranil Gupta (Indy)

CONCURRENCY CONTROL

Lecture A

RPCs

Why RPCs

- RPC = Remote Procedure Call
- Proposed by Birrell and Nelson in 1984
- Important abstraction for processes to call functions in other processes
- Allows code reuse
- Implemented and used in most distributed systems, including cloud computing systems
- Counterpart in Object-based settings is called RMI (Remote Method Invocation)

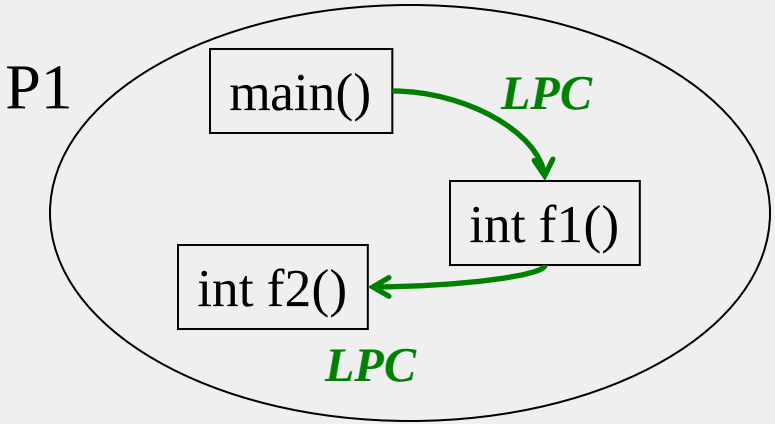
Local Procedure Call (LPC)

- Call from one function to another function within the same process
  - Uses stack to pass arguments and return values
  - Accesses objects via pointers (e.g., C) or by reference (e.g., Java)
- LPC has ***exactly-once*** semantics
  - If process is alive, called function executed exactly once

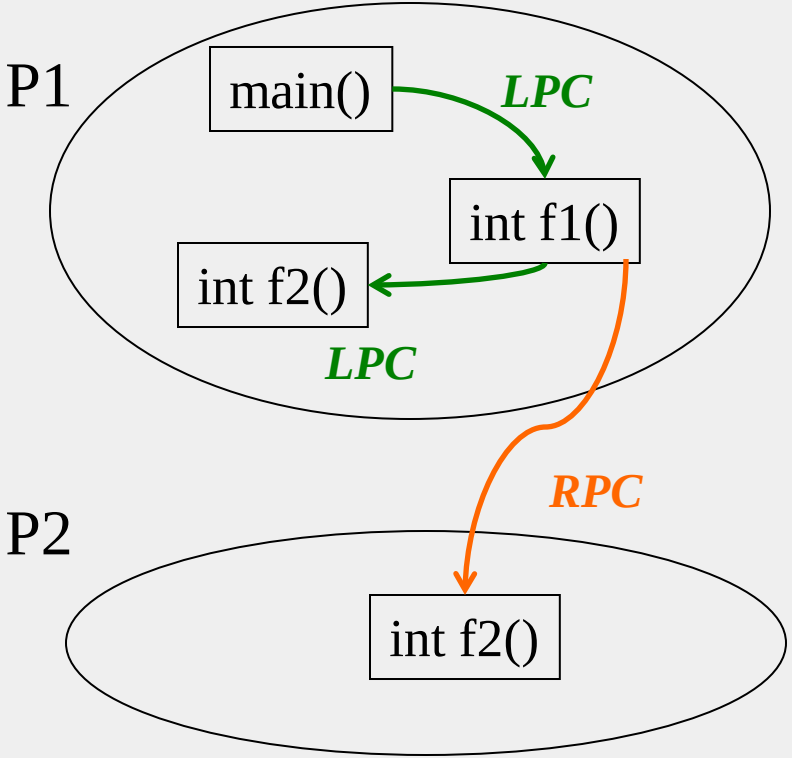
Remote Procedure Call

- Call from one function to another function, where caller and callee function reside in different processes
  - Function call crosses a process boundary
  - Accesses objects via global references
    - Can’t use pointers across processes since a reference address in process P1 may point to a different object in another process P2
    - E.g., Object address = IP + port + object number
- Similarly, RMI (Remote Method Invocation) in Object-based settings

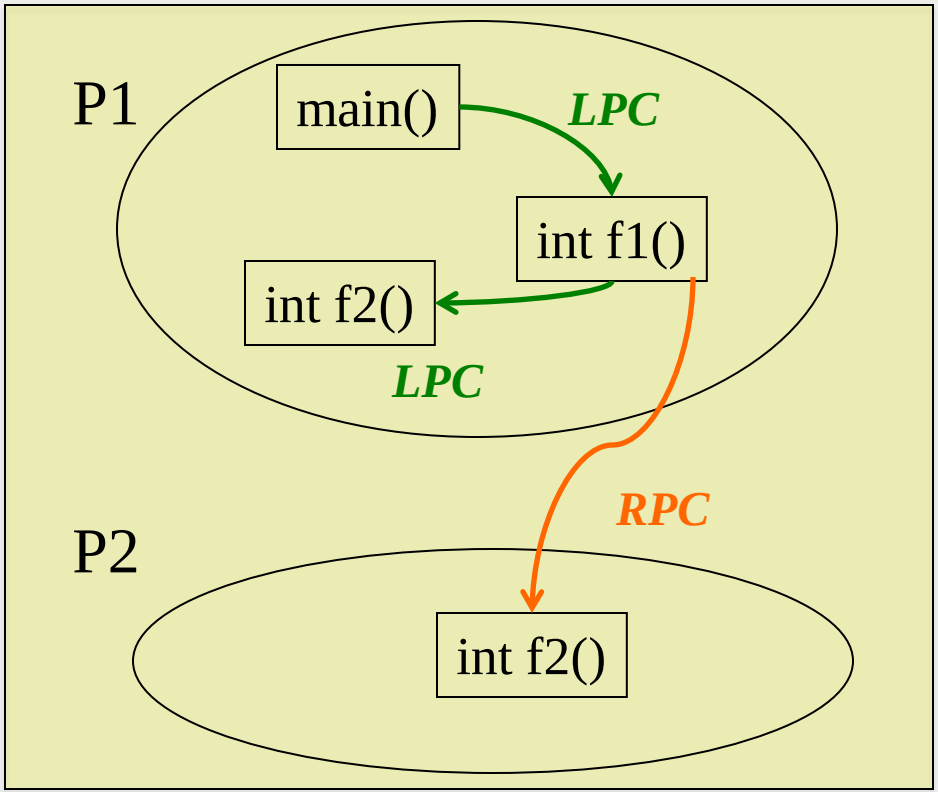
LPCs



RPCs

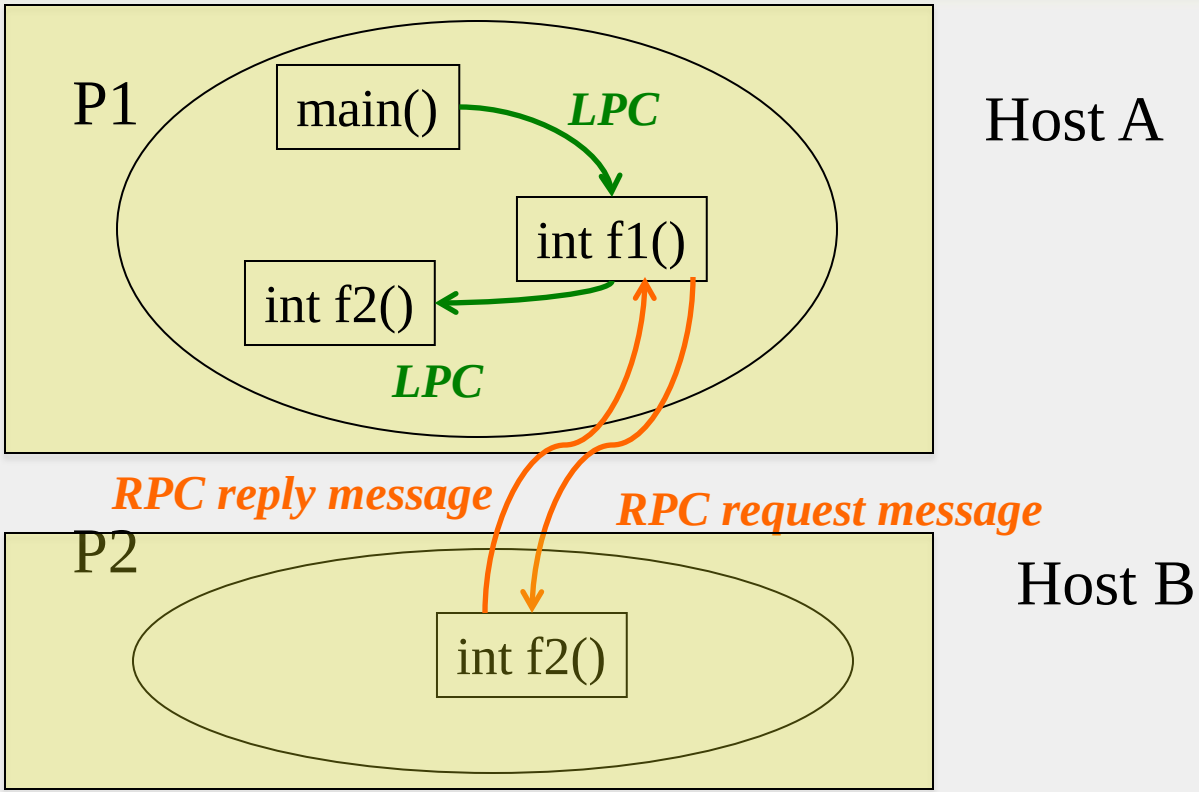


RPCs



Host A

RPCs


P1
main()
LPC
int f1()
int f2()
LPC
RPC reply message
RPC request message
P2
int f2()
Host A
Host B

RPC Call Semantics

- Under failures, hard to guarantee exactly-once semantics
- Function may not be executed if
  - Request (call) message is dropped
  - Reply (return) message is dropped
  - Called process fails before executing called function
  - Called process fails after executing called function
  - Hard for caller to distinguish these cases
- Function may be executed multiple times if
  - Request (call) message is duplicated

Implementing RPC Call Semantics

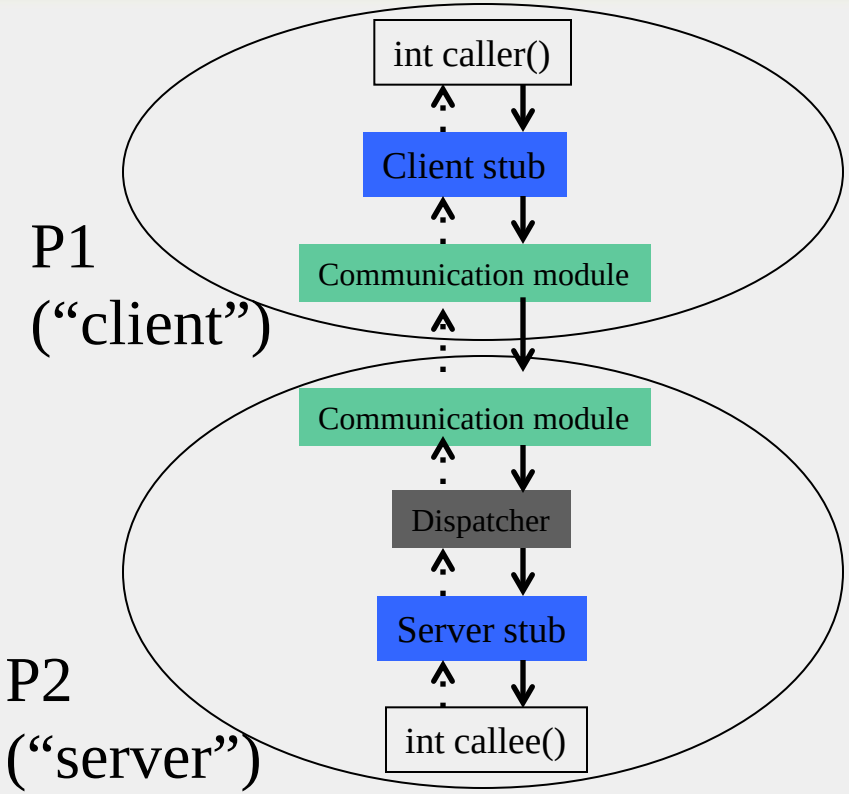
- Possible semantics
  - At most once semantics (e.g., Java RMI)
  - At least once semantics (e.g., Sun RPC)
  - Maybe, i.e., best-effort (e.g., CORBA)

Retransmit request	Filter duplicate requests	Re-execute function or retransmit reply	RPC Semantics
Yes	No	Re-execute	At least once
Yes	Yes	Retransmit	At most once
No	NA	NA	Maybe

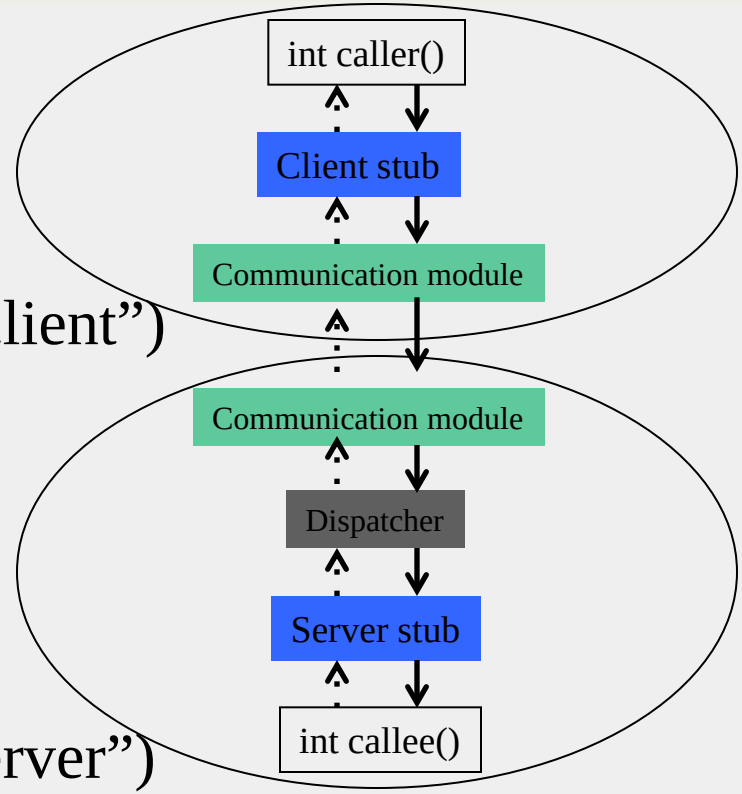
Idempotent Operations

- Idempotent operations are those that can be repeated multiple times, without any side effects
- Examples (x is server-side variable)
  - x=1;
  - x=(argument) y;
- Non-examples
  - x=x+1;
  - x=x*2;
- Idempotent operations can be used with at-least-once semantics

Implementing RPCs



RPC Components



Client

- **Client stub**: has same function signature as callee()
  - Allows same caller() code to be used for LPC and RPC
- **Communication Module**: Forwards requests and replies to appropriate hosts

Server

- **Dispatcher**: Selects which server stub to forward request to
- **Server stub**: calls callee(), allows it to return a value

Generating Code

- Programmer only writes code for caller function and callee function
- Code for remaining components all **generated automatically** from function signatures (or object interfaces in Object-based languages)
  - E.g., Sun RPC system: Sun XDR interface representation fed into rpcgen compiler
- These components together part of a Middleware system
  - E.g., CORBA (Common Object Request Brokerage Architecture)
  - E.g., Sun RPC
  - E.g., Java RMI

Marshalling

- Different architectures use different ways of representing data
  - **Big endian**: Hex 12-AC-33 stored with 12 in lowest address, then AC in next higher address, then 33 in highest address
    - IBM z, System 360
  - **Little endian**: Hex 12-AC-33 stored with 33 in lowest address, then AC in next higher address, then 12
    - Intel
- Caller (and callee) process uses its own *platform-dependent* way of storing data
- Middleware has a common data representation (CDR)
  - *Platform-independent*

Marshalling (2)

- Middleware has a common data representation (CDR)
  - Platform-independent
- Caller process converts arguments into CDR format
  - Called “Marshalling”
- Callee process extracts arguments from message into its own platform-dependent format
  - Called “Unmarshalling”
- Return values are marshalled on callee process and unmarshalled at caller process

Next

- Now that we know RPCs, we can use them as a building block to understand transactions