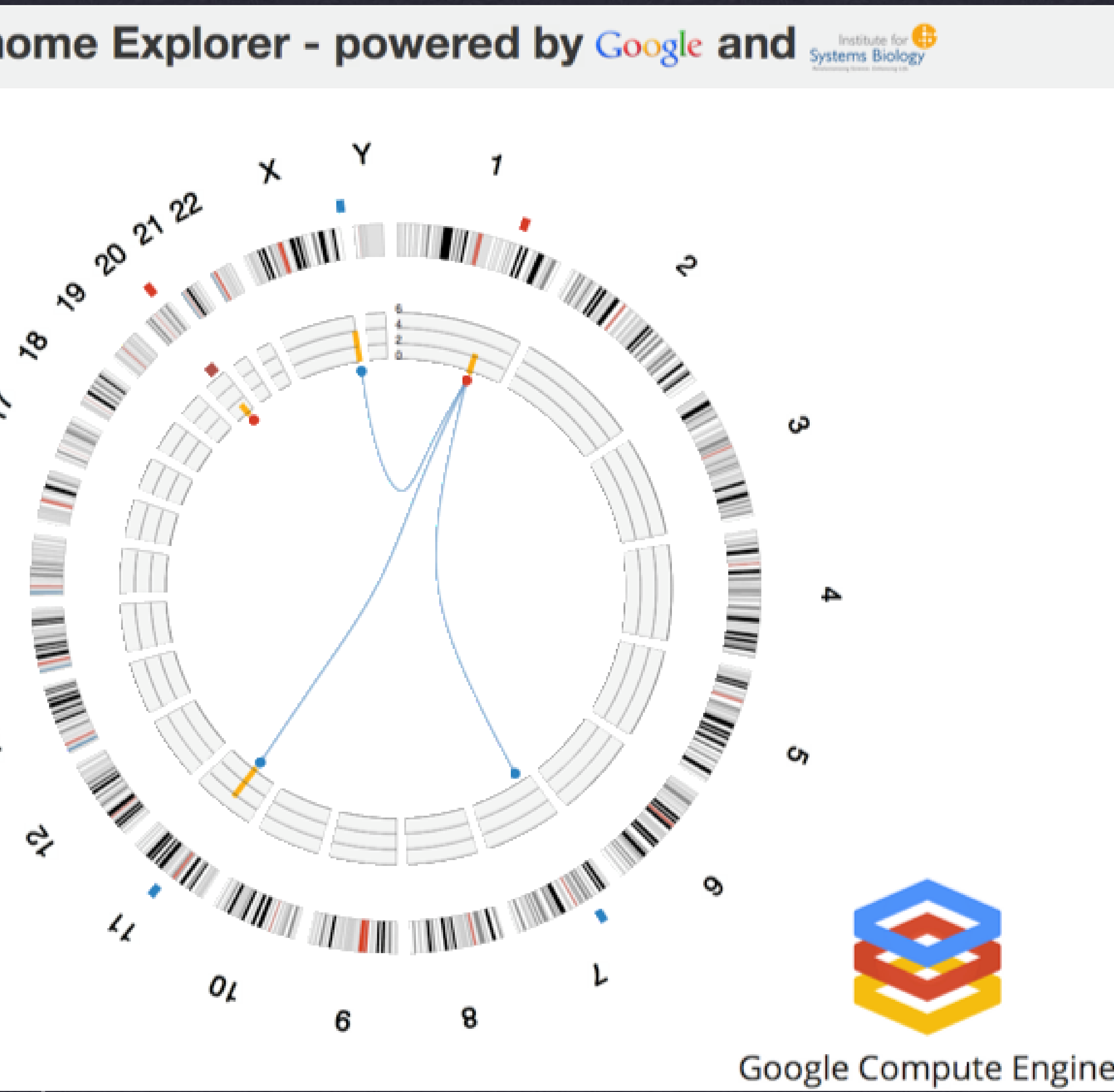
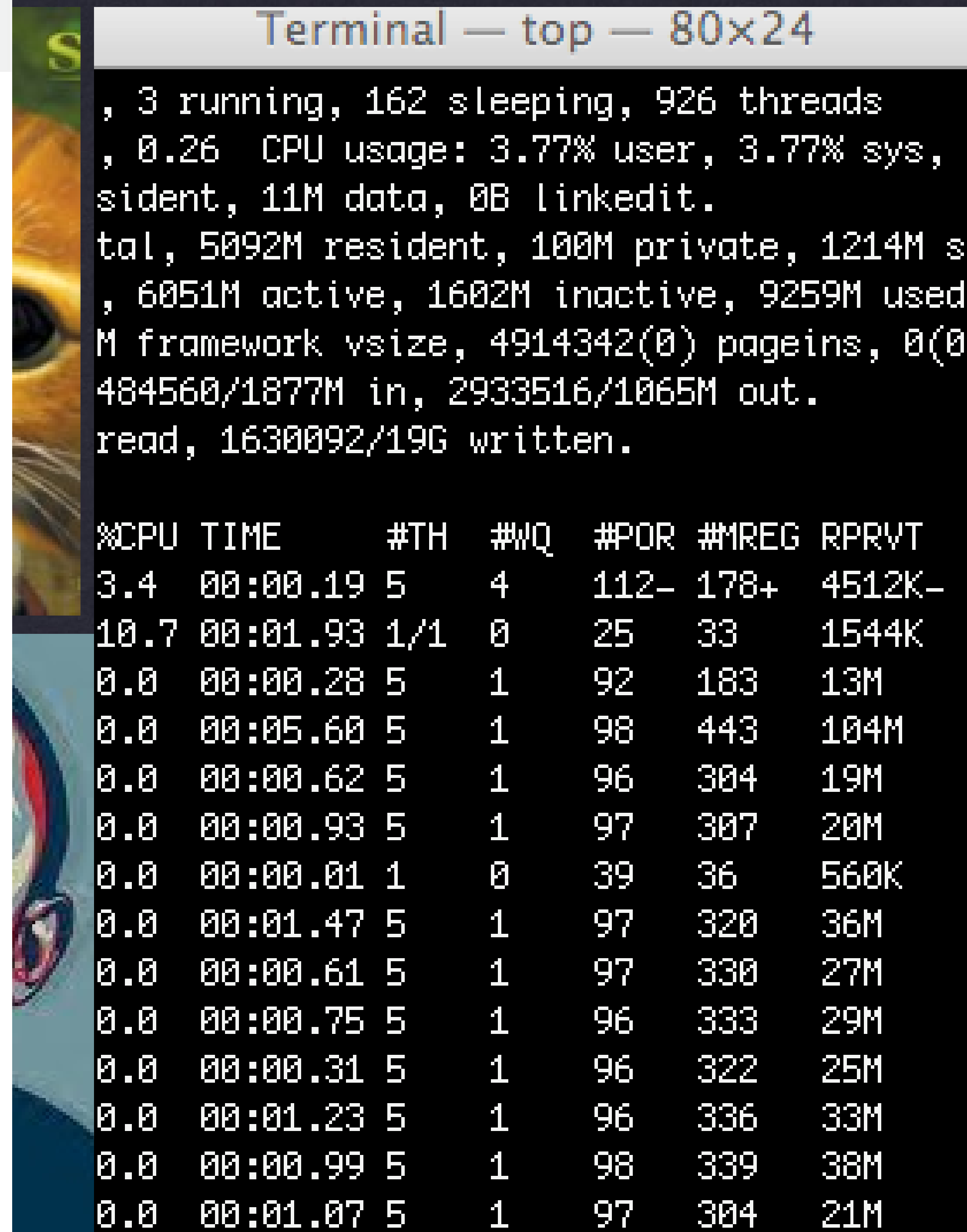




distributed systems

Last week ...

- Synchronisation
- Distributed Clocks

Revision Quiz

This week ...

- Today’s lecture will be all about remote operations

Remote Procedure Call – Readings

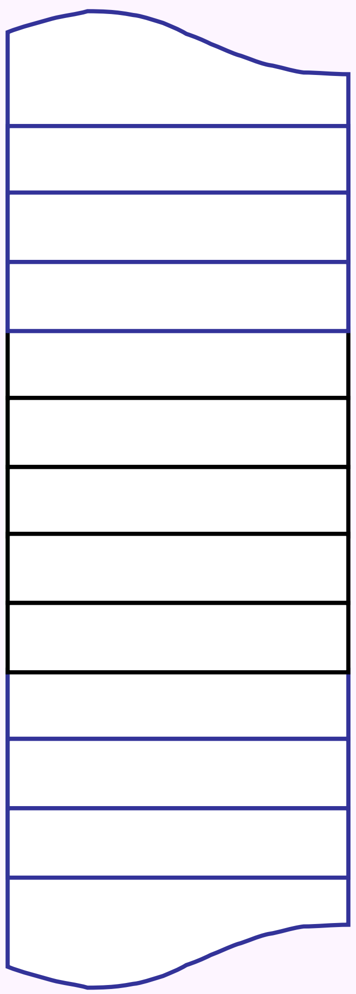
- Systems such as Java's RMI (Remote Method Invocation) are part of Remote Procedure Call (RPC) systems:
  - Java RMI is an RPC facility for objects, for calls of the form: O.m(params…)
  - The original RPC systems are formulated in terms of procedures, with calls of the form: P(params…)
- References
  - Birrell and Nelson, "Implementing Remote Procedure Call", ACM Transactions on Computer Systems, Vol. 2, No. 1, February 1984.

Theme

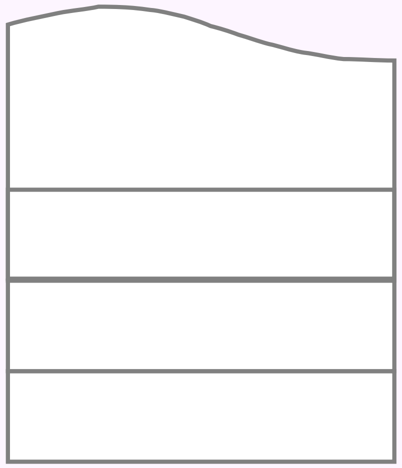
- *A distributed software system is one in which the provision of a service may occur on a different computer to the call requesting that service.*
  - A call that crosses machine boundaries is referred to as a ***remote operation***.
  - This section of the course is concerned with how to provide calls through remote operations.
  - The focus is on the *special characteristics* of calls in a distributed environment.

Local calls

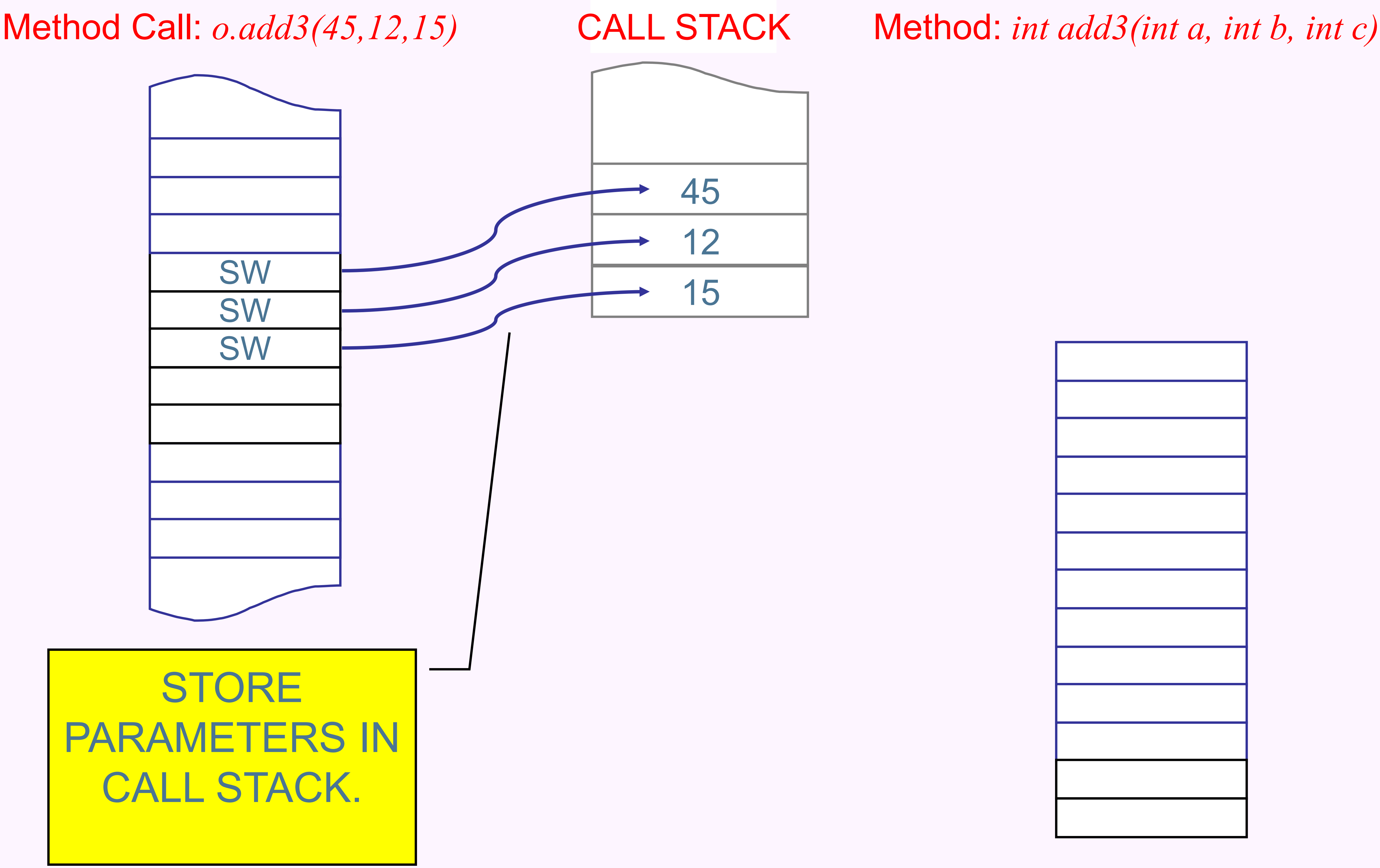
Method Call: *o.add3(45,12,15)*

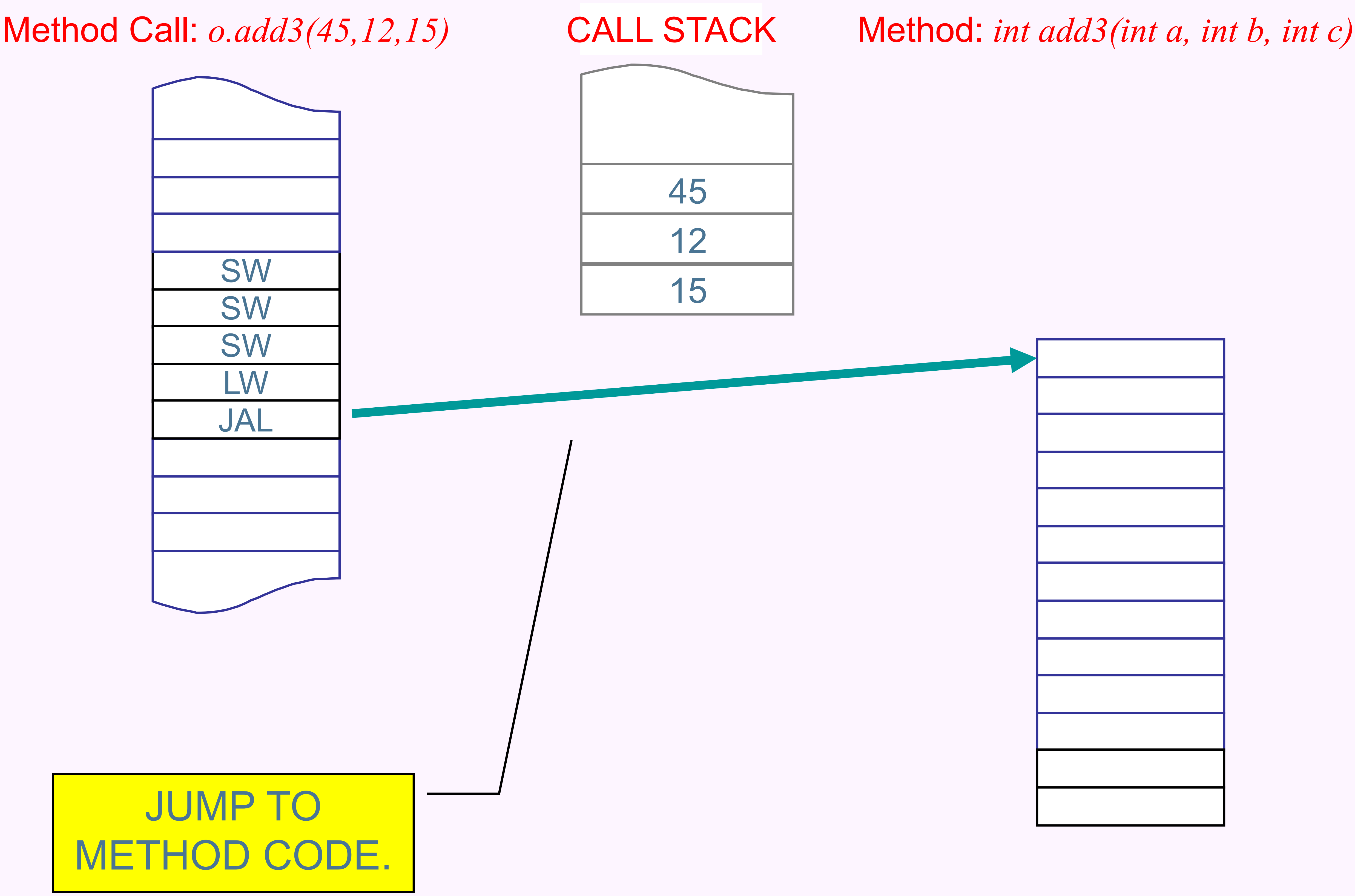


CALL STACK

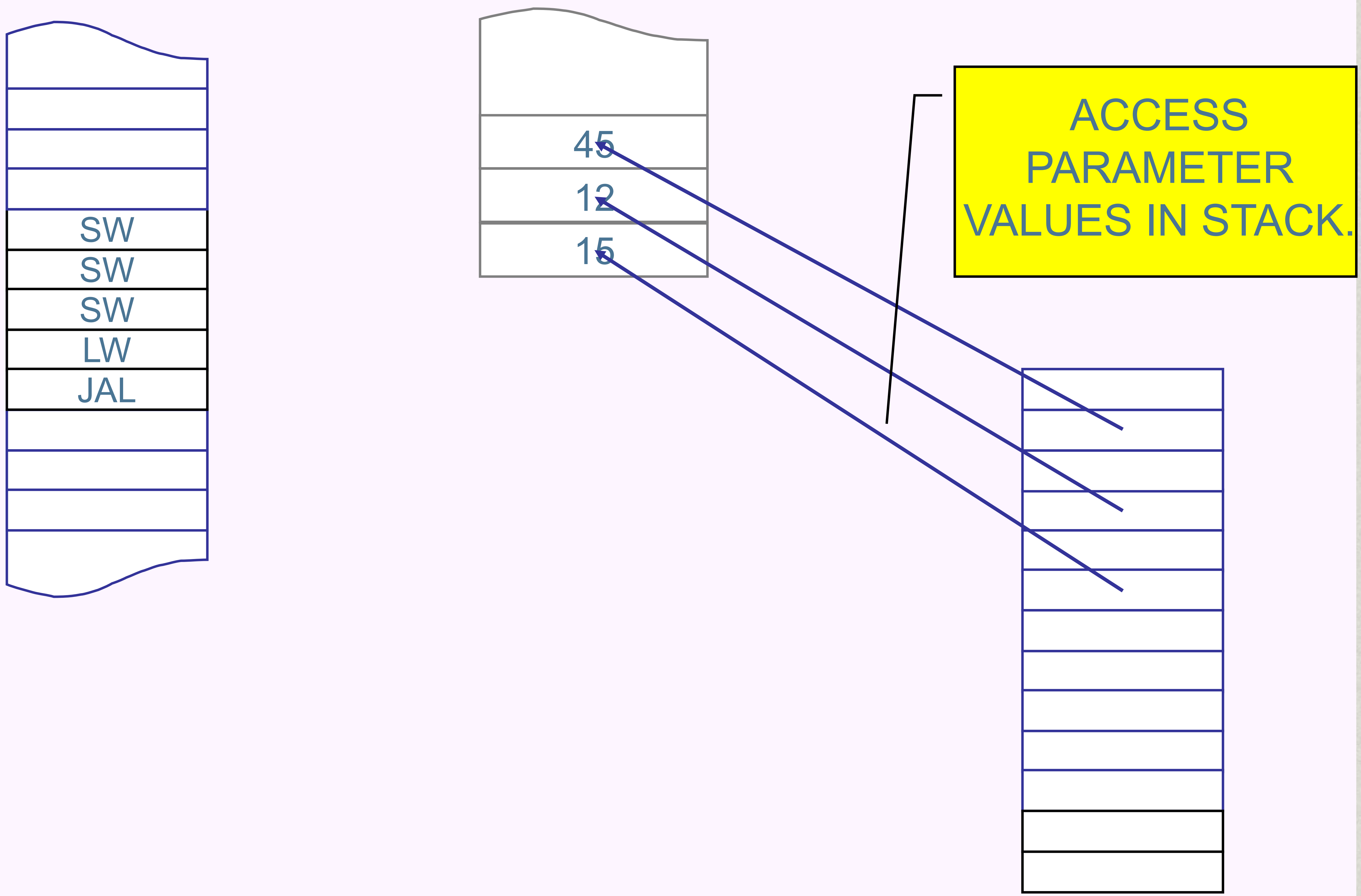


Method: *int add3(int a, int b, int c)*

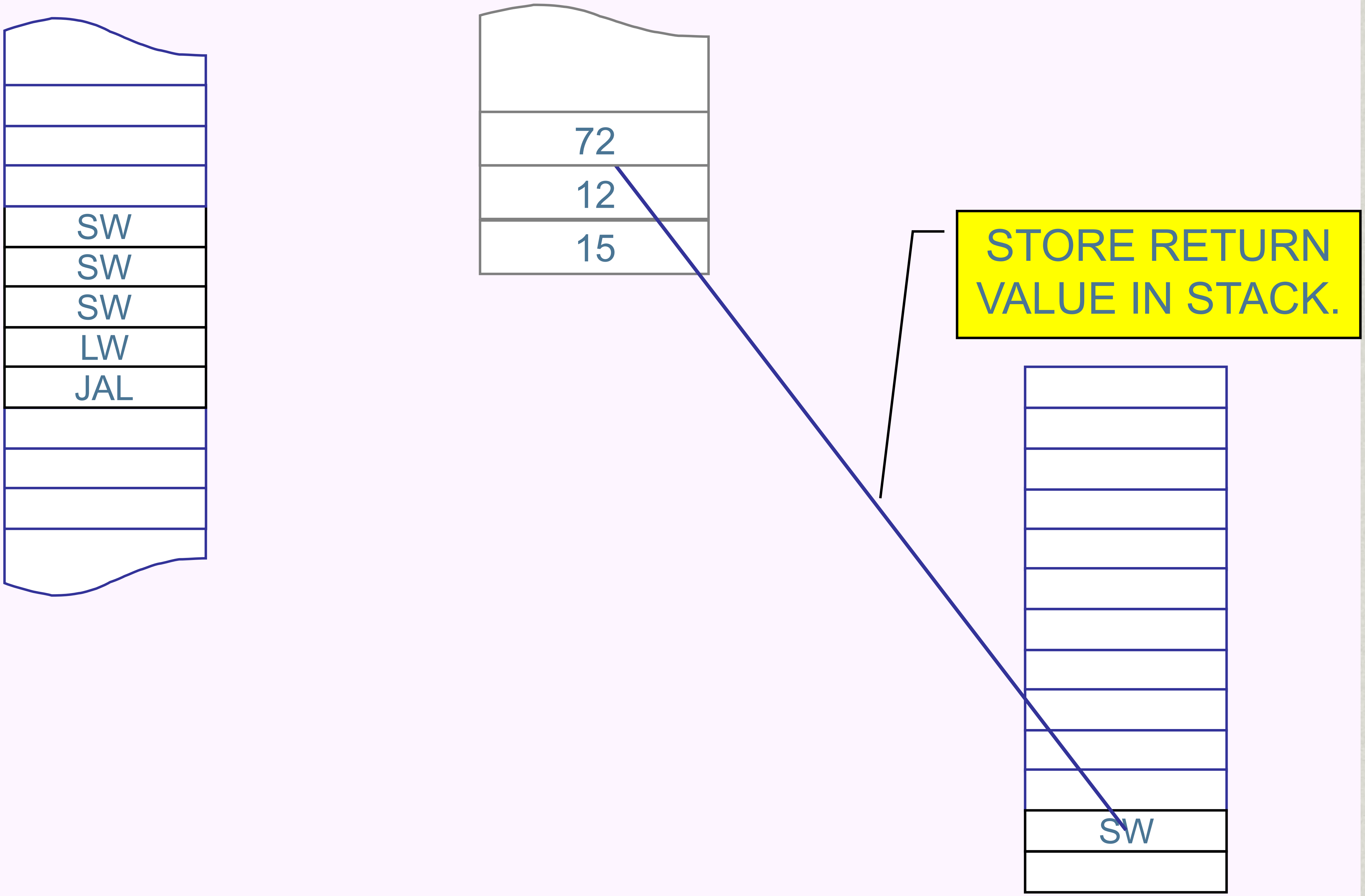

Method Call: *o.add3(45,12,15)*
CALL STACK
Method: *int add3(int a, int b, int c)*
SW
SW
SW
45
12
15
STORE PARAMETERS IN CALL STACK.


Method Call: *o.add3(45,12,15)*
CALL STACK
Method: *int add3(int a, int b, int c)*
SW
SW
SW
LW
JAL
45
12
15
JUMP TO METHOD CODE.

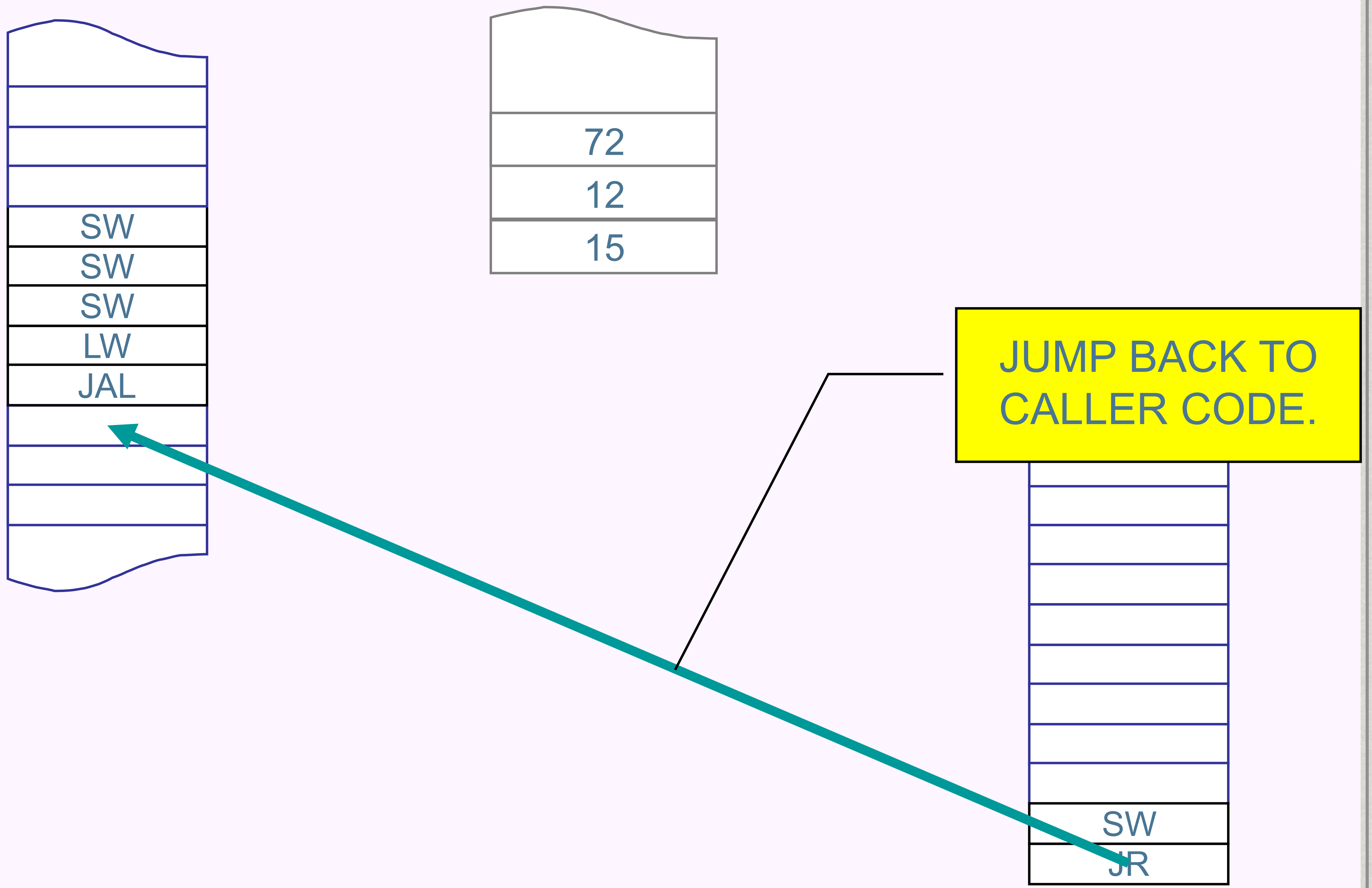
Method Call: *o.add3(45,12,15)* CALL STACK Method: *int add3(int a, int b, int c)*

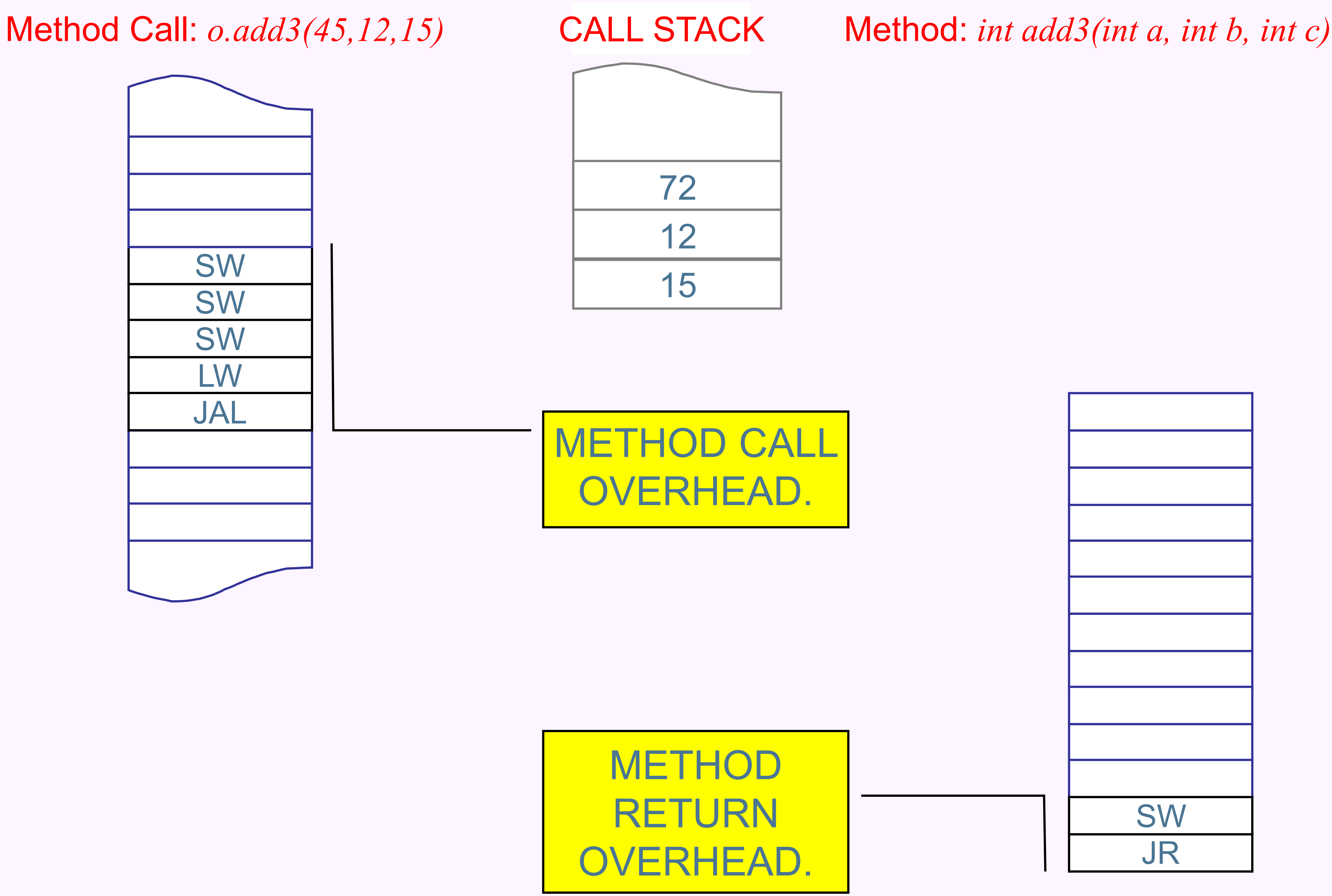


Method Call: *o.add3(45,12,15)* CALL STACK Method: *int add3(int a, int b, int c)*



Method Call: *o.add3(45,12,15)* CALL STACK Method: *int add3(int a, int b, int c)*




Method Call: o.add3(45,12,15)
CALL STACK
Method: int add3(int a, int b, int c)
72
12
15
SW
SW
SW
LW
JAL
METHOD CALL OVERHEAD.
METHOD RETURN OVERHEAD.
SW
JR

Questions

1. How much time is spent in call overhead for a local call?

2. How much time is spent in return overhead for a local call?

3. How do parameter values get communicated from caller to service in a local call?

4. How does the result value get communicated from service to caller in a local call?

Questions (cont’d)

5. If the caller of a local call is written in Java, then what language is the service written in?

6. If a service is written in Java, and that service is called locally, then what language is the caller written in?

The Big Question

What about remote calls?

Questions Reprised

1. How much time is spent in call overhead for a **remote** call?

2. How much time is spent in return overhead for a **remote** call?

3. How do parameter values get communicated from caller to service in a **remote** call?

4. How does the result value get communicated from service to caller in a **remote** call?

Questions Reprised

5. If the caller of a **remote** call is written in Java, then what language is the service written in?

6. If a service is written in Java, and that service is called **remotely** , then what language is the caller written in?

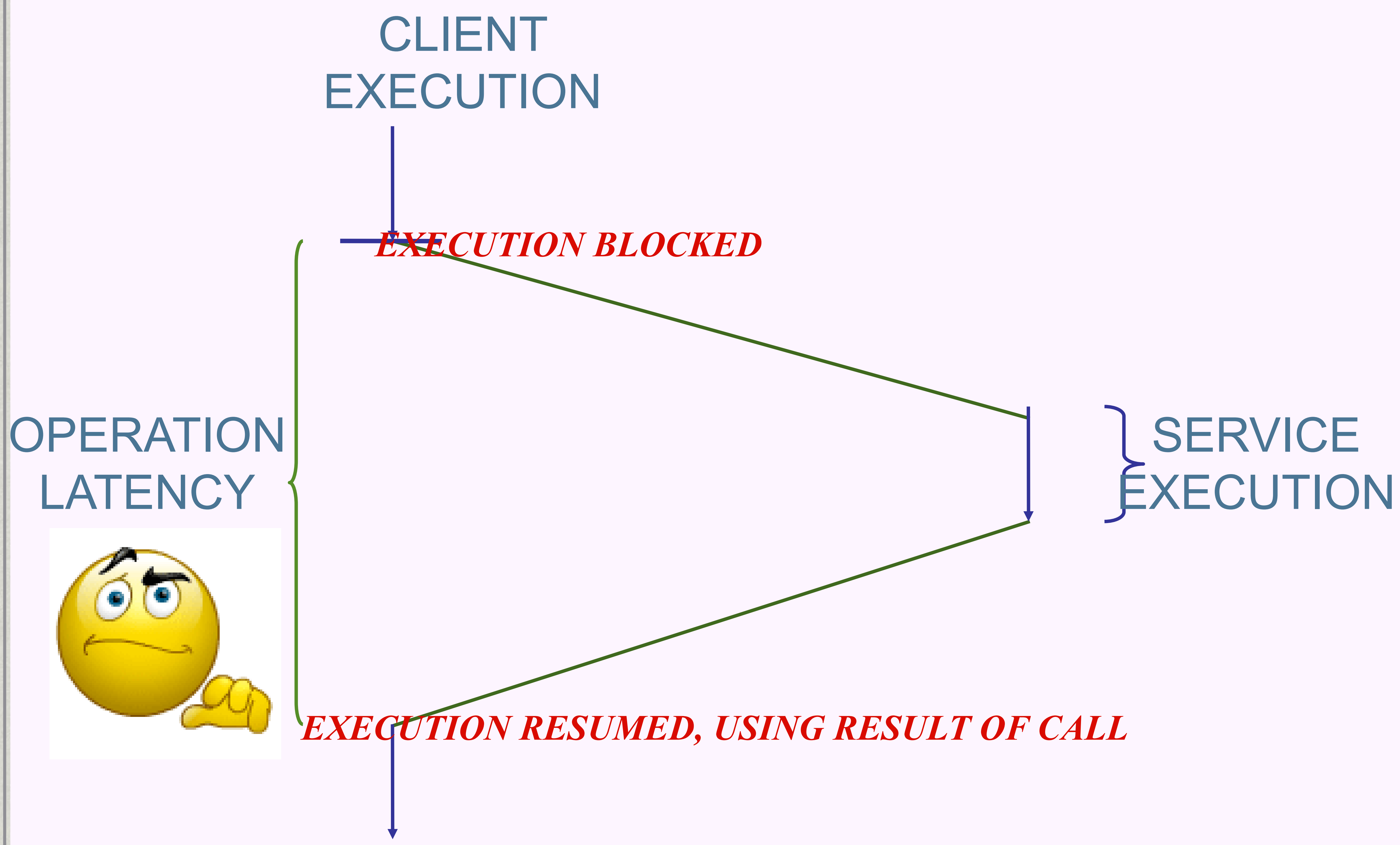
When it comes to RPC..

We need to think about

- How can we cope with *remote operation latencies*?
- How can we provide parameter and result transmission over a network?
- How can we deal with *linguistic heterogeneity*?

Recall …

- Remote operation latency is the time a client is waiting for the result after starting the call:
  - … thumb twiddling time … – the client is doing nothing during this time.
- Even on a local area network, during most of this time the server isn’t doing anything either.


CLIENT EXECUTION
EXECUTION BLOCKED
OPERATION LATENCY
SERVICE EXECUTION
EXECUTION RESUMED, USING RESULT OF CALL

Recall …

- Remote operation latency is the time a client is waiting for the result after starting the call:
  - … thumb twiddling time … – the client is doing nothing during this time.
- Even on a local area network, during much of this time the server isn’t doing anything either.
- On a wide area network, latencies are so great that the vast majority of time is spent with both client and server doing nothing.
- Client or server?

CLIENT
EXECUTION

THINK ABOUT HOW TO DEAL WITH LATENCY …

EVERY TIME IT MAKES A CALL, THE CLIENT PAYS A HEAVY PRICE IN WASTED TIME

EXECUTION BLOCKED

OPERATION
LATENCY

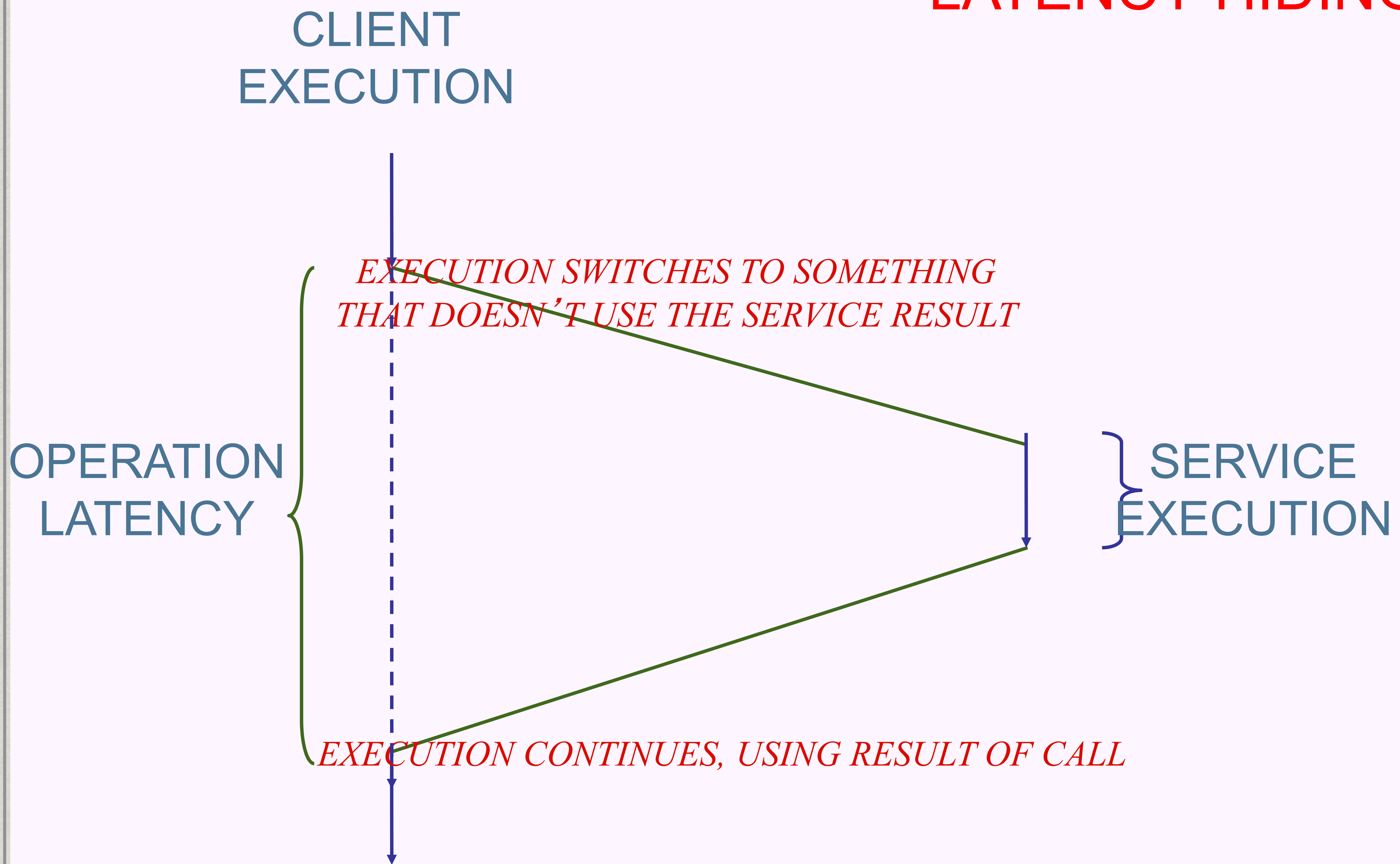
SERVICE
EXECUTION

EXECUTION RESUMED, USING RESULT OF CALL

Dealing with Latency

- Two basic strategies:
  - Latency hiding.
  - Latency reduction.
- … and many variations of these general strategies.

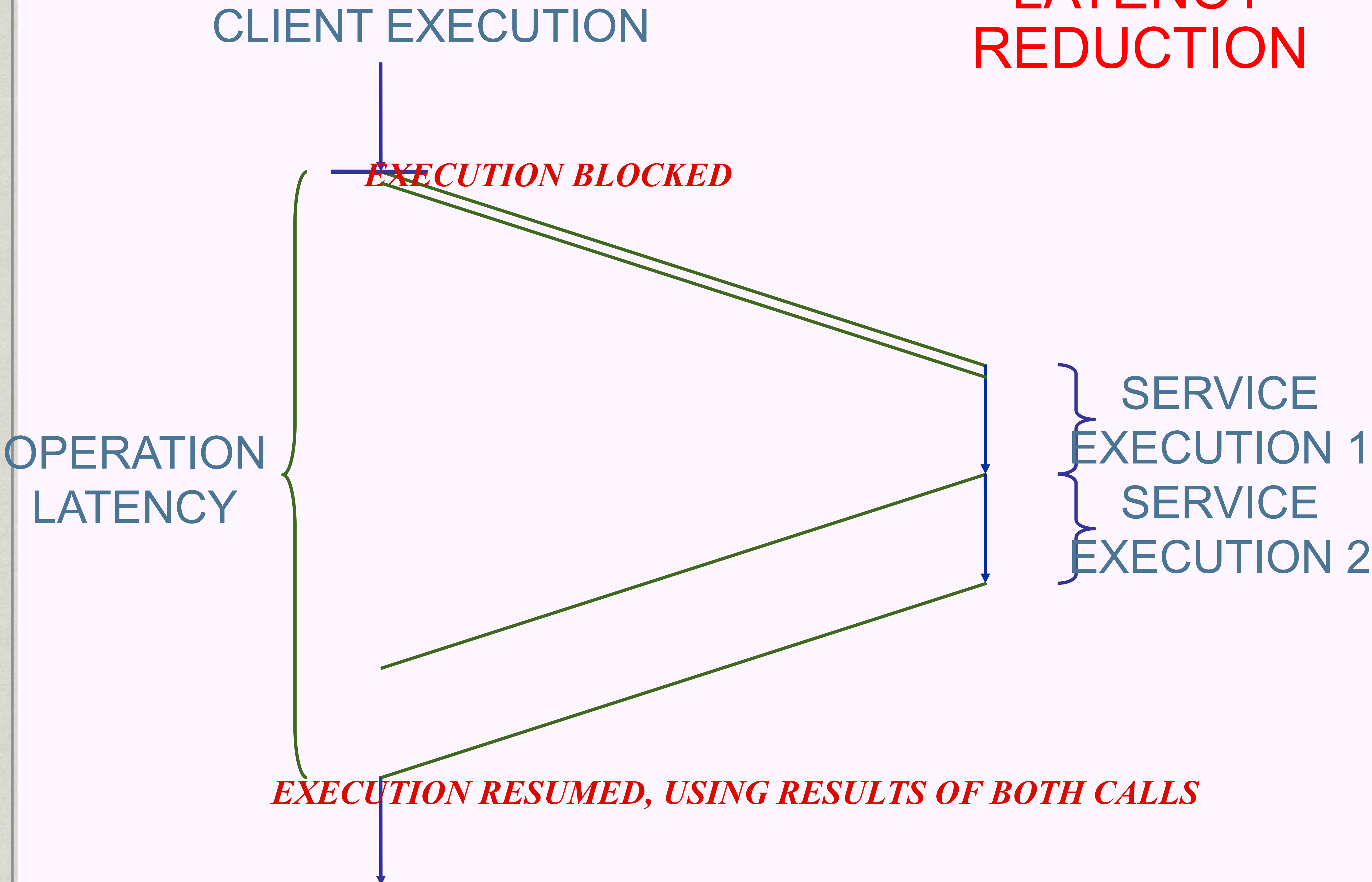
LATENCY HIDING



Latency Hiding

- Tries to avoid wasting the time during which the client is waiting for a call by doing useful work instead.
- Useful work can’t depend on the result of the call since it hasn’t arrived yet.
  - Program clients carefully
  - Determine dependencies – still might have to wait as this might not be accurate

LATENCY REDUCTION



Latency Reduction

- Tries to reduce the **average latency** of remote calls by running multiple calls at the same time.
- This is only possible if calls can be sent off before the results of all previous calls are available.

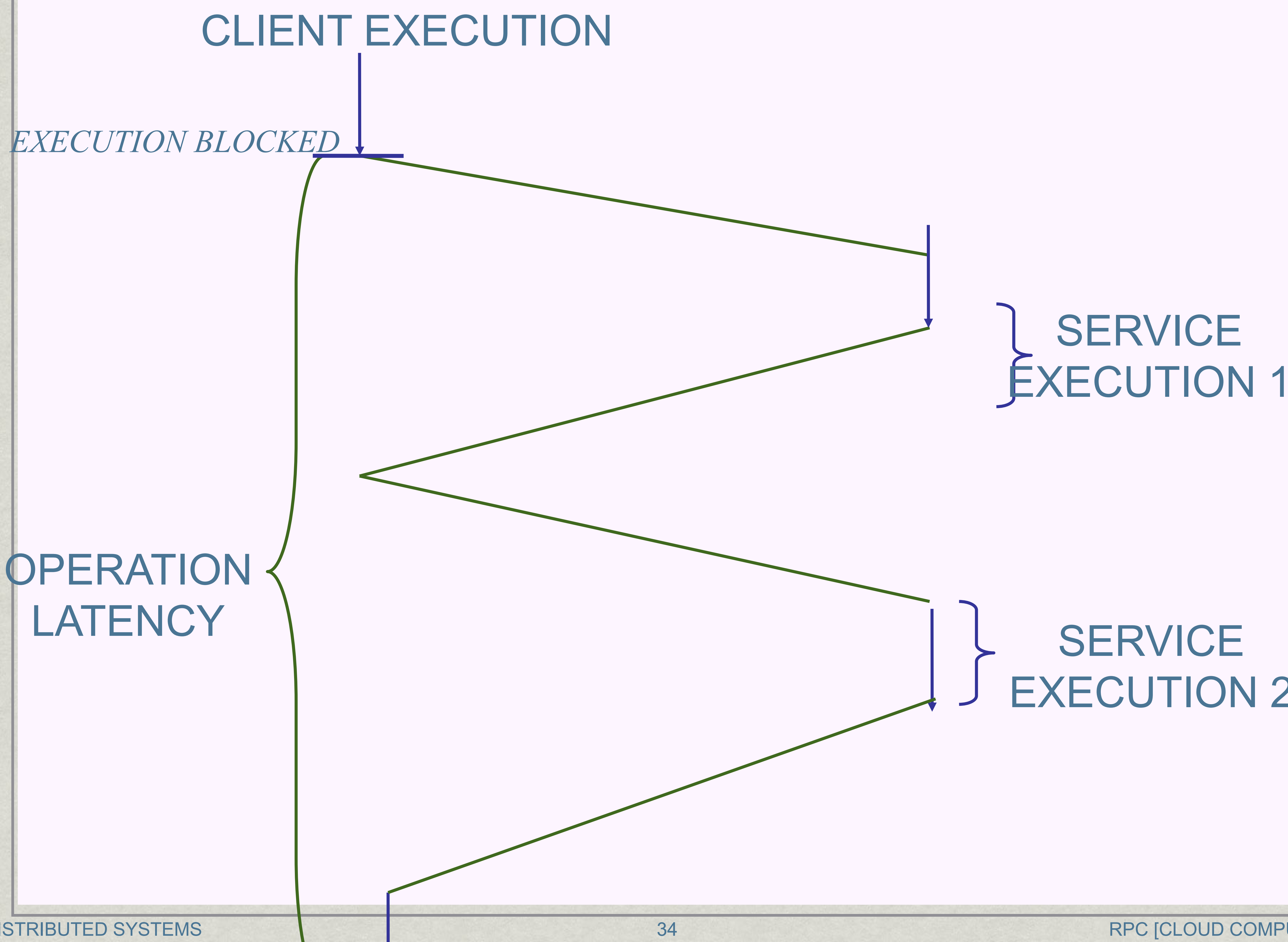
Average Latency

- Total time taken for client divided by the number of operations

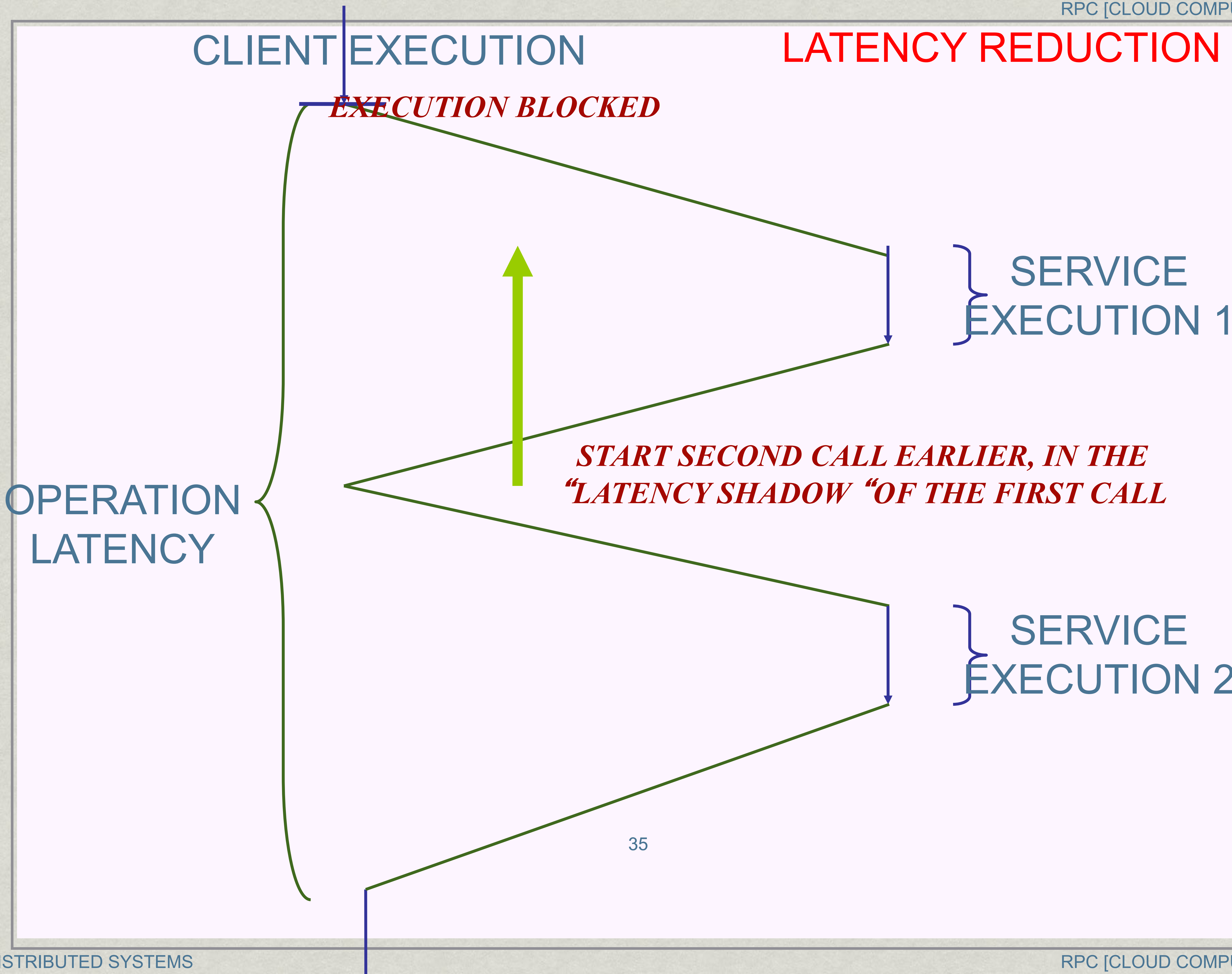
- vs

- Average time taken for each operation individually

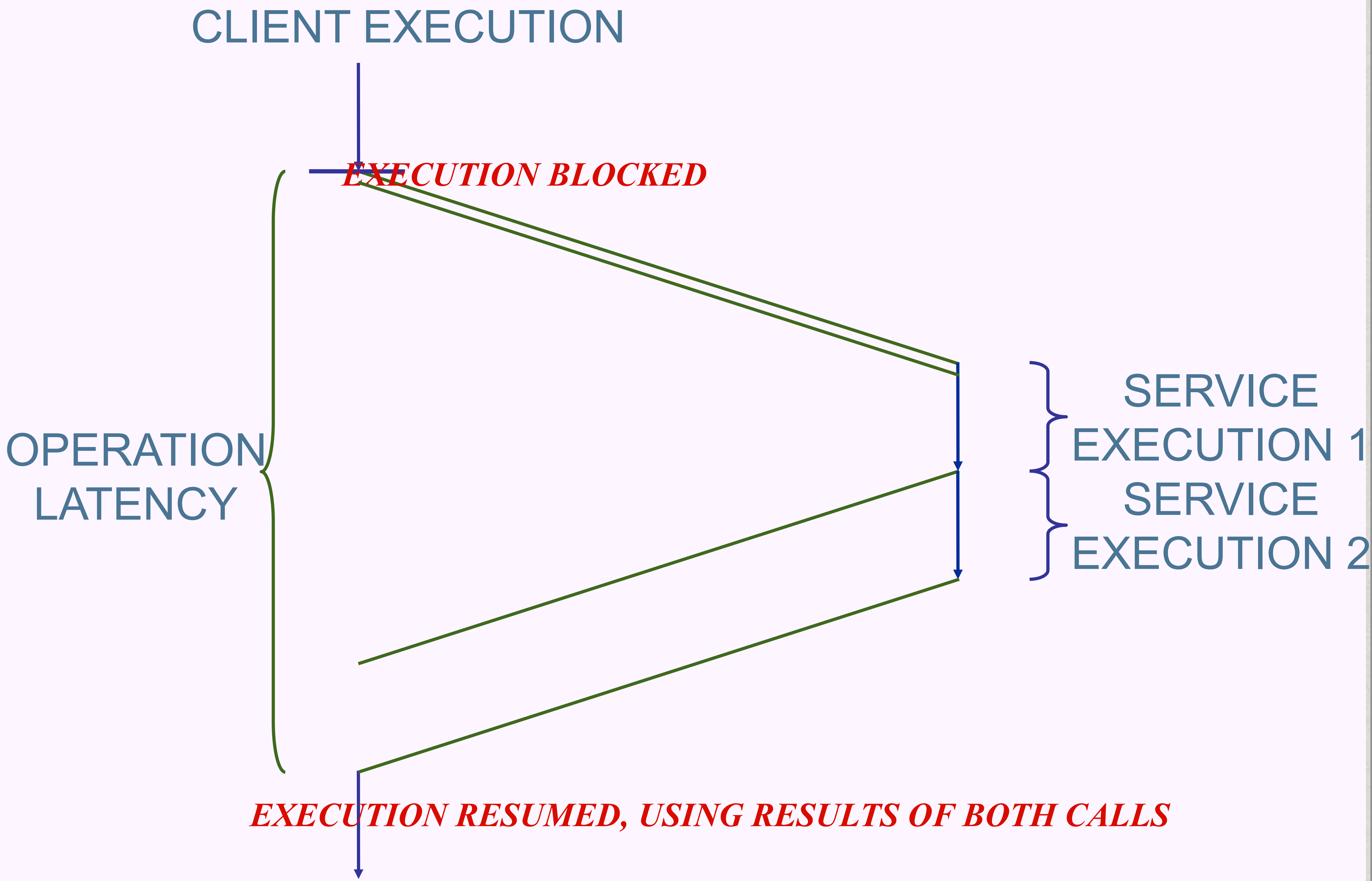
LATENCY REDUCTION



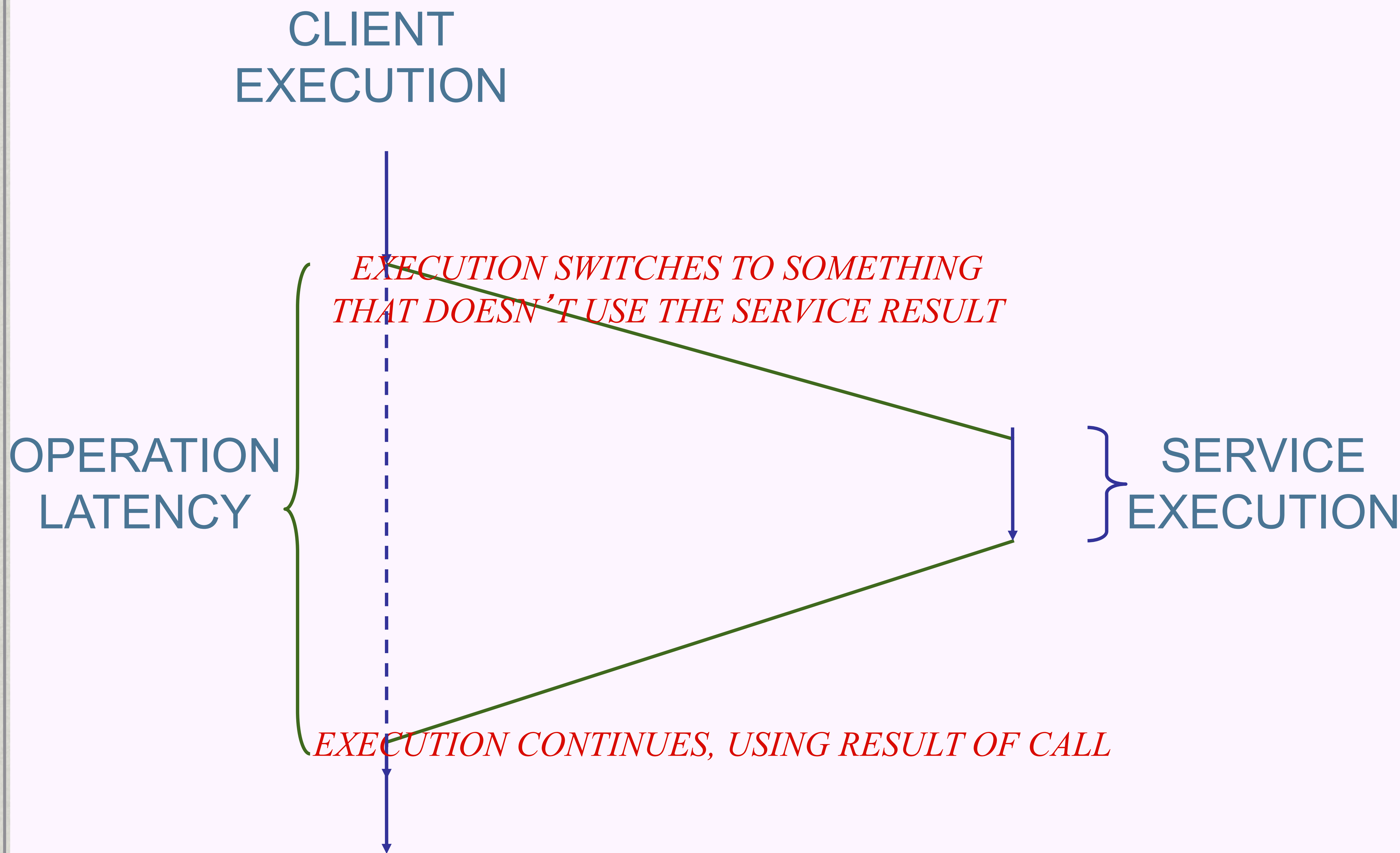
LATENCY REDUCTION



LATENCY REDUCTION



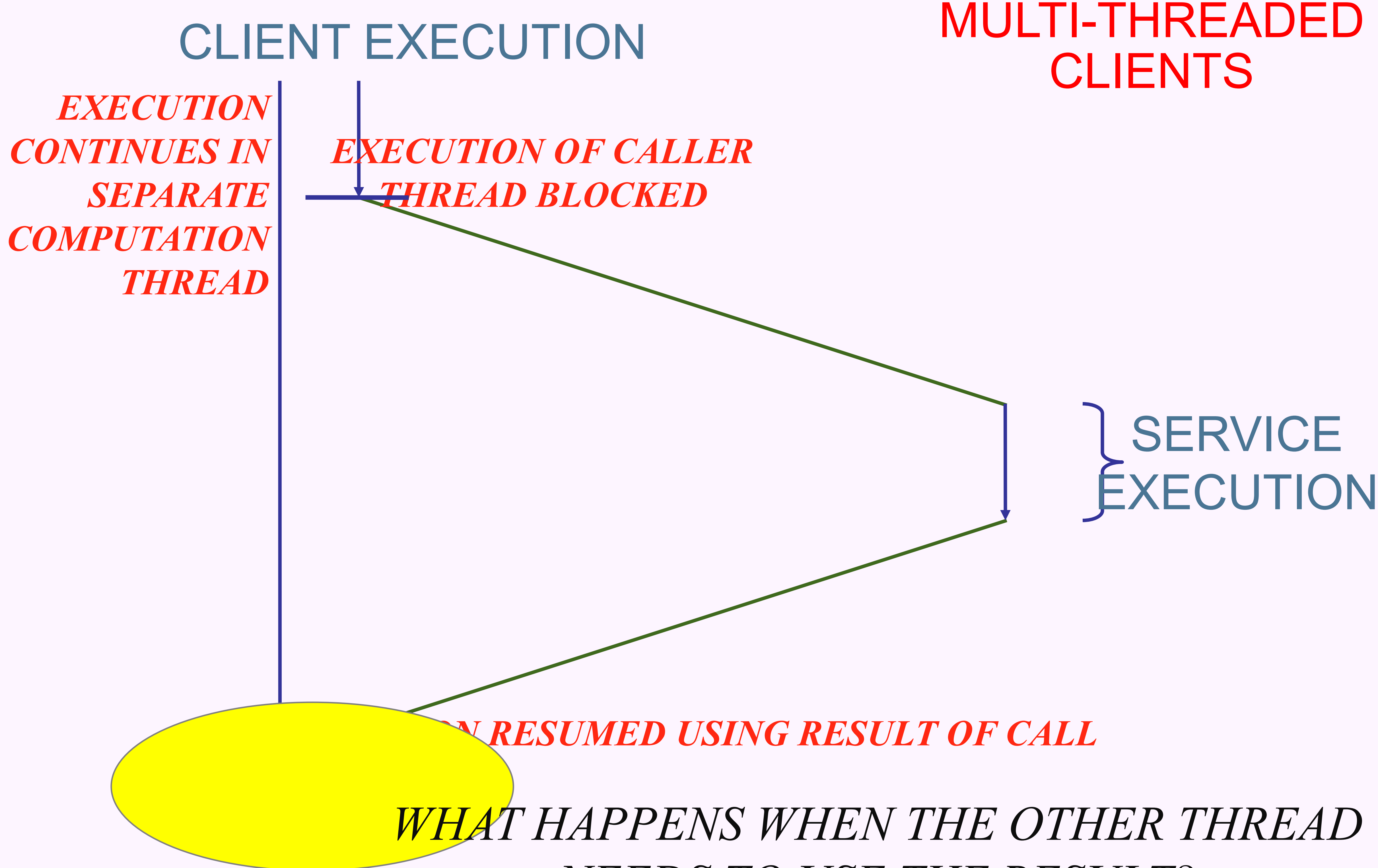
LATENCY HIDING



Issues - Latency Hiding

- Finding useful work for the client to do
    - Application-specific
- Reply synchronisation:
    - How does the client find out that the result has arrived?
    - …so it can stop doing other work and make use of the result.
- Reply matching:
    - How does the client work out which request a given reply corresponds to?

LATENCY HIDING MULTI-THREADED CLIENTS



WHAT HAPPENS WHEN THE OTHER THREAD NEEDS TO USE THE RESULT?

Latency Hiding with Multi-threaded Clients

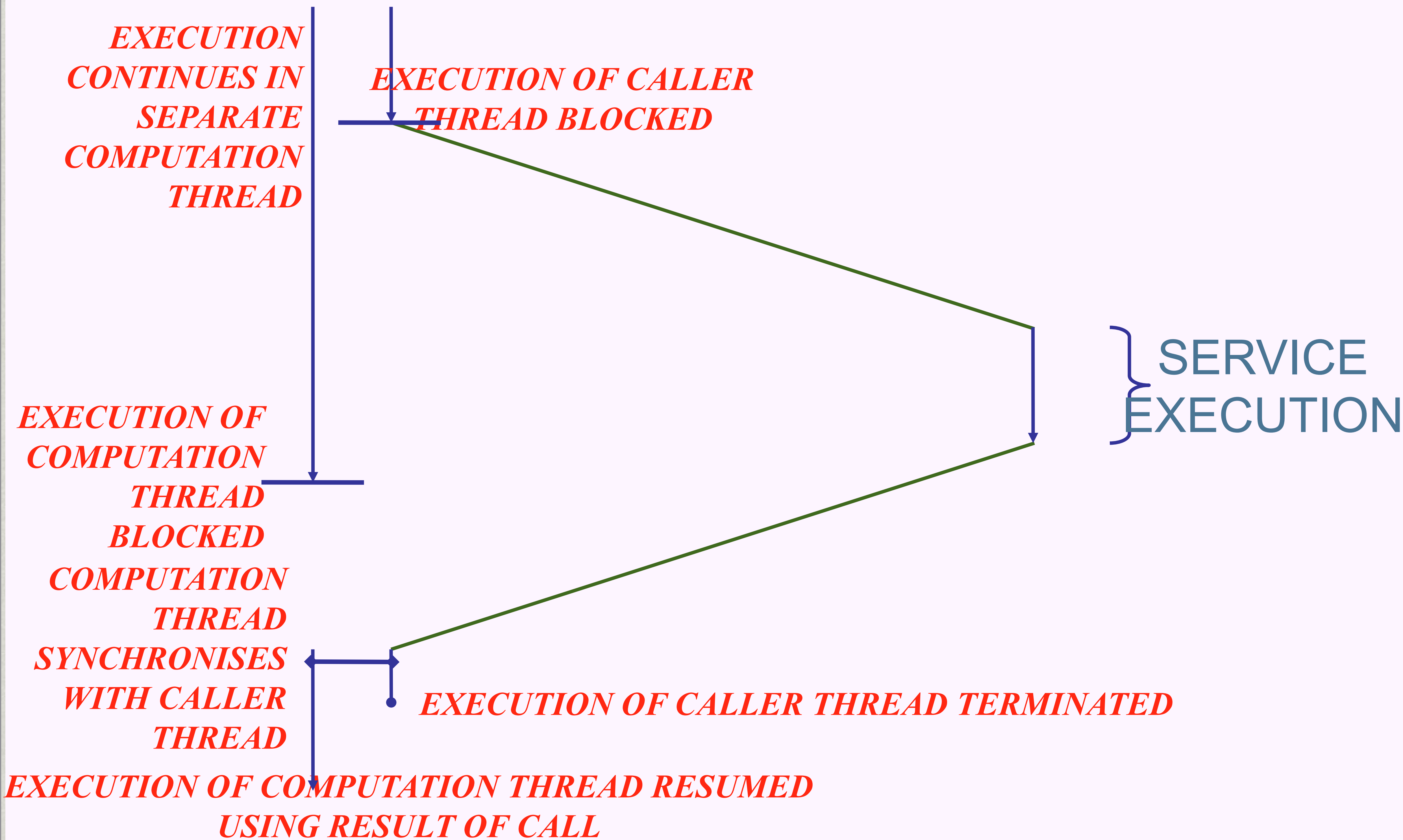
- Good choice if the computation thread(s) never need to know about the result:
    - I.e. no reply synchronisation required.
    - Very unlikely situation in practice!
- Reasonable choice if the caller thread is doing some kind of “background task” like sending a print job:
    - Once the call has completed, the caller thread places the result in a shared data structure (e.g. a queue).
    - Periodically, the computation thread inspects the queue, processing any queued results it finds.

Latency Hiding with Multi-threaded Clients

- Most common approach is for the ***caller thread to be started by a/the computation thread***, for the purpose of doing the call:
  - the computation thread needs the result at some later stage.
  - Will have to wait for the caller thread to finish the call, then synchronise with that thread and exchange information.

THREADS FOR LATENCY HIDING

CLIENT EXECUTION



Latency Hiding with Multi-threaded Clients

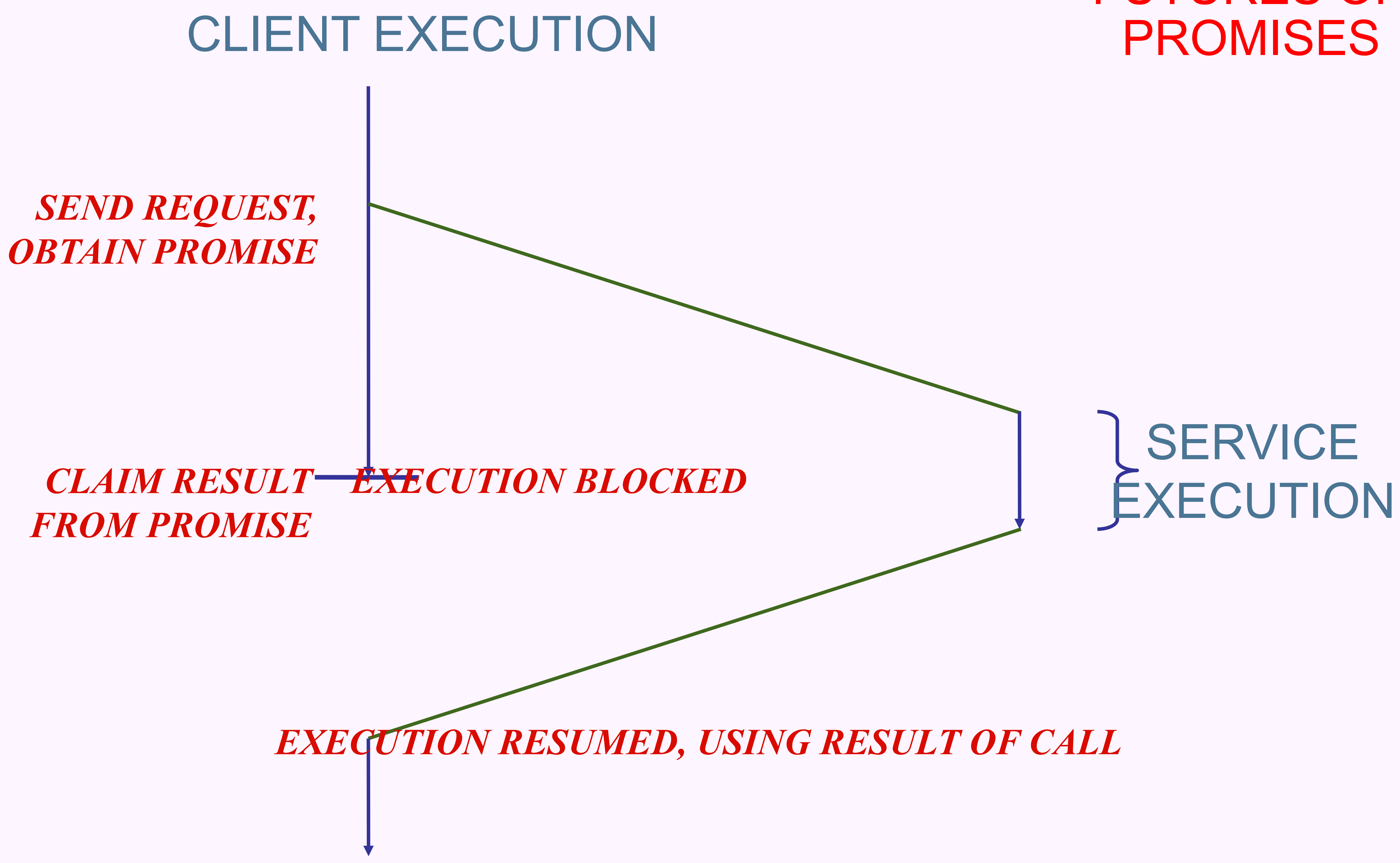
- Have to wait for the caller thread to finish the call, then synchronise with that thread and exchange information.
    - Can use multiple caller threads to run more than one call in parallel -> have to do reply matching
- Problems?
    - Reply matching:
        - Identify that we need to wait
        - Figure out what the result is
    - Must create and manage multiple threads
        - Synchronization, concurrency, barriers, semaphores, mutual exclusion, the three heads of cerberus …

Can we do better than this?

Futures or promises

- We get a promise back when we make the invocation
  - This is something that represents the result, but isn’t the result.
- When we need to use the result, we “claim” it which forces us to wait until the execution is finished.

FUTURES OR PROMISES



Latency Hiding with Promises and Futures

- *Reply synchronisation is fairly easy*:
  - The claim operation blocks the client until the reply arrives.
  - If the reply arrives before the claim, the client is not notified but this usually doesn't matter – client hasn't claimed, so it obviously doesn't need the result.
  - ***Some systems have a non-blocking operation on promises/futures to test reply arrival.***

Implicit vs Explicit

- Implicit
    - Any use of the future automatically obtains its value, as if it were an ordinary reference
    - try to use the result and the system will claim it if necessary
    - no need to change code
    - difficult to implement
- Explicit
    - must call the function to obtain the value
    - `java.util.concurrent.Future.get`

Latency Hiding with Promises and Futures

- Avoids unnecessary multi-threaded programming.
- Can easily send off multiple requests:
  - Each call returns a distinct promise/future.
  - Reply matching is easy since the **promises/futures are distinct**.
  - Provides a good basis for latency reduction.

Activity: Cloud Computing & Data centers

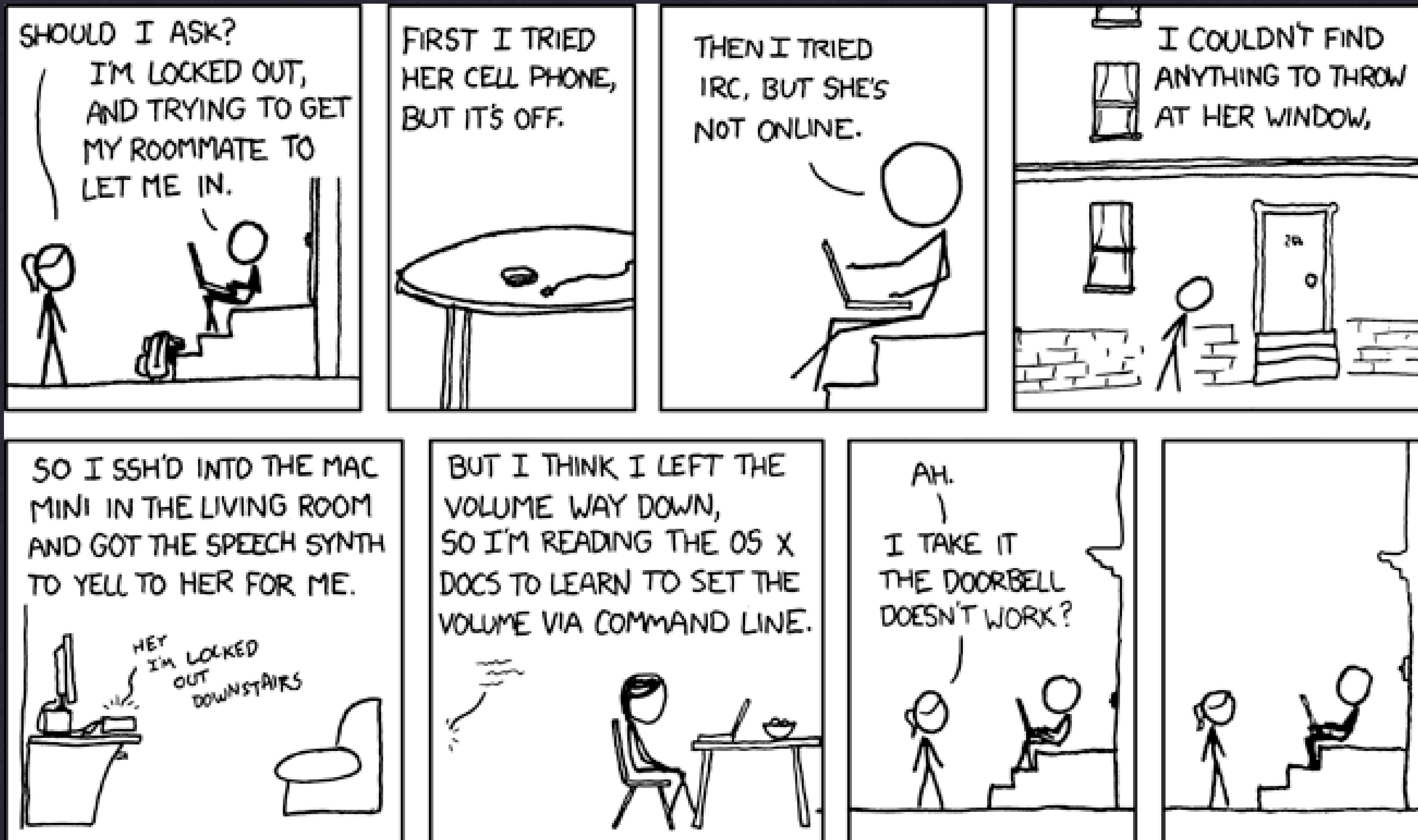
- Cloud computing and data centers
- Why?
- http://www.youtube.com/watch?v=Hl5o-n9UrFk

What?

- 63 million render hours and 117 terabytes of data
    - 7187 years
- 200 high-performance HP Z800 Workstations to design everything in the film
- HP ProLiant BL460 blade technology powered five different server render farms geographically dispersed across the U.S. and India
- HP Cloud Services rendered 8 million hours

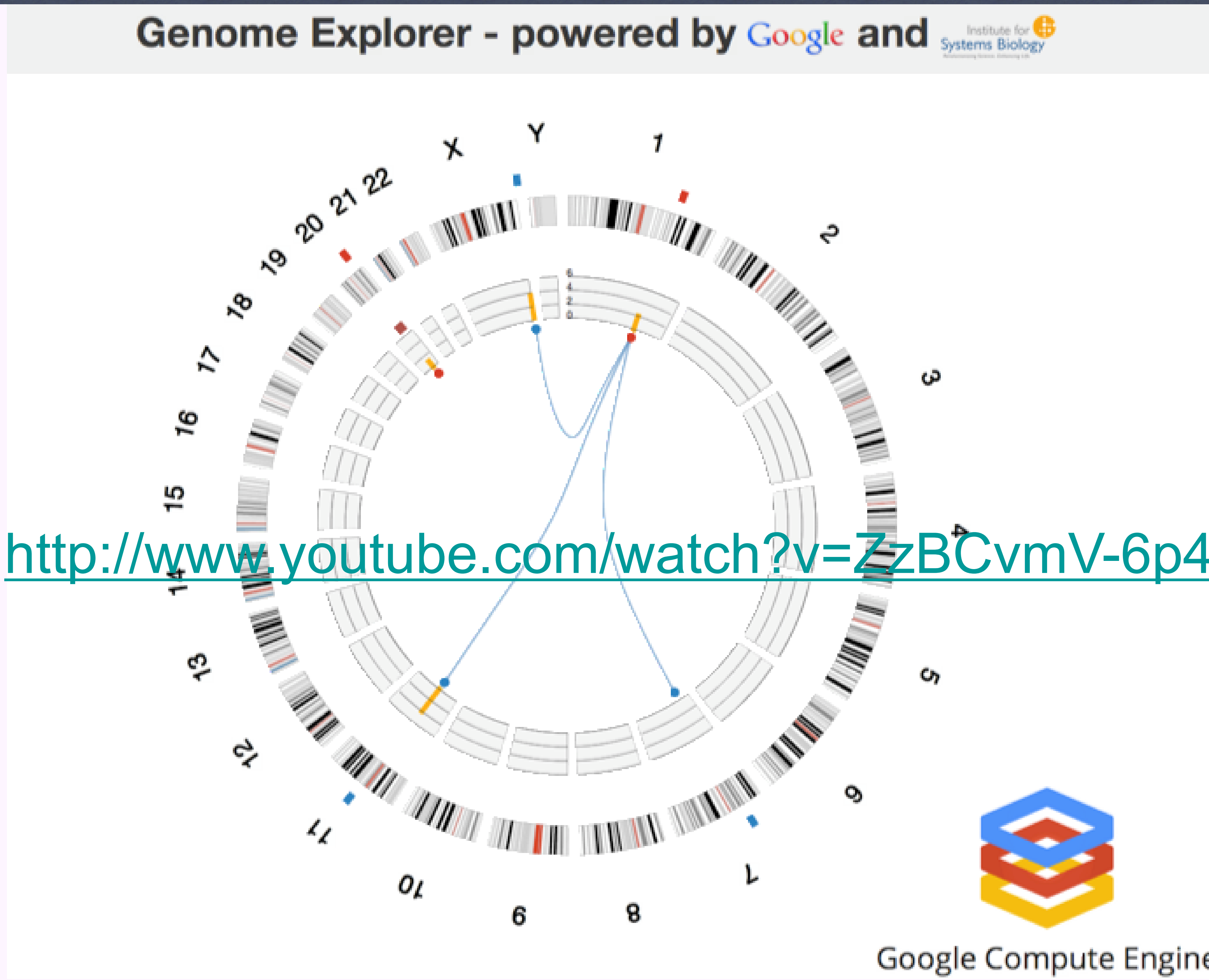
How?

- http://www.youtube.com/watch?feature=player_detailpage&v=YQERVf9ibzY



Systems talk (XKCD 530)

Google Compute Engine

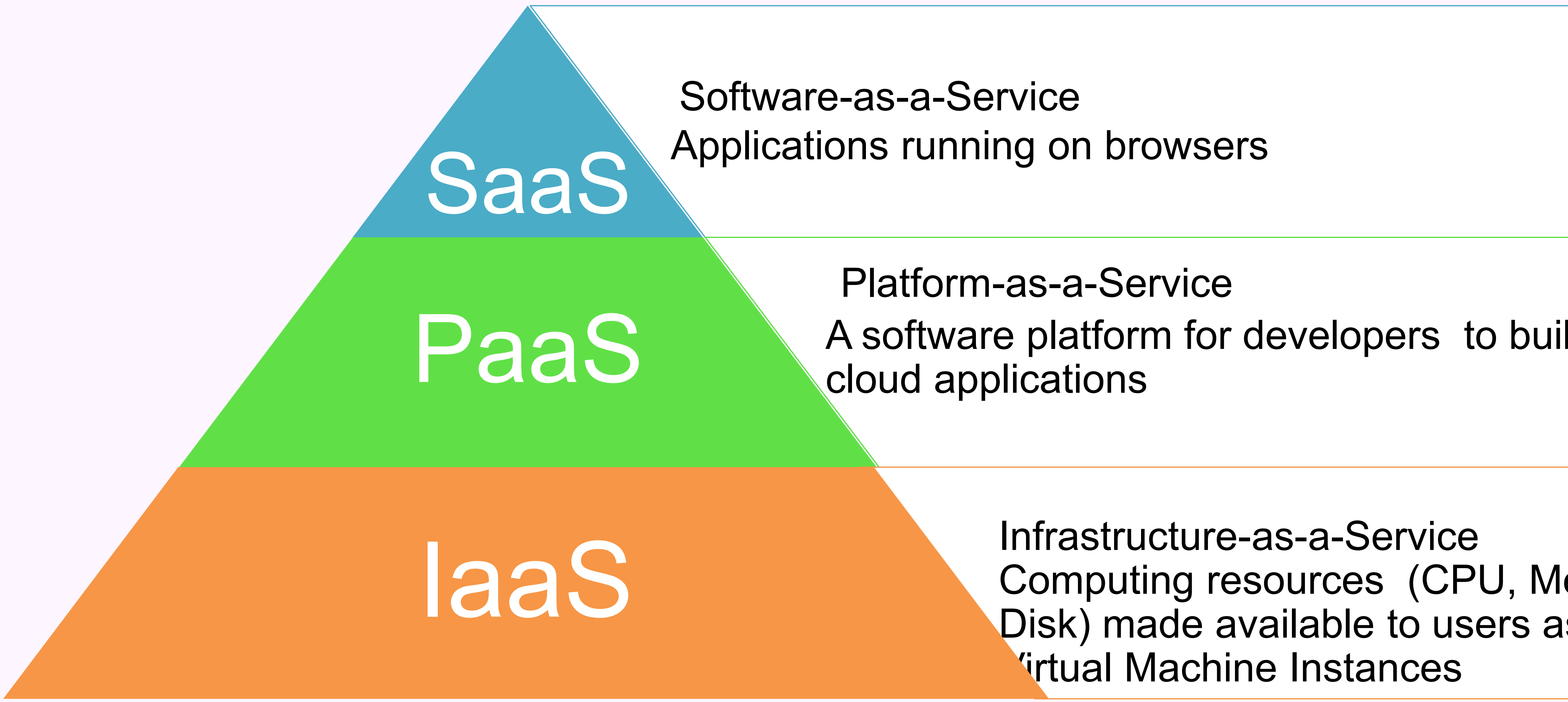


http://www.youtube.com/watch?v=ZzBCvmV-6p4

Key Cloud Characteristics

- On-demand self-service through a service portal
- Rapid elasticity – time to market / fast deployment
- Pay per use
- Ubiquitous access
- Location-independent resource pooling

Cloud Service Models



Pros and Cons of Service Models

• Service Models	Pros	Cons
• Traditional	highest flexibility	time-to-market
• IaaS •	scalability, no hw procure	privacy
• PaaS •	DB, Frameworks, middleware ready	vendor lock in
• SaaS	time-to-market	lowest flexibility

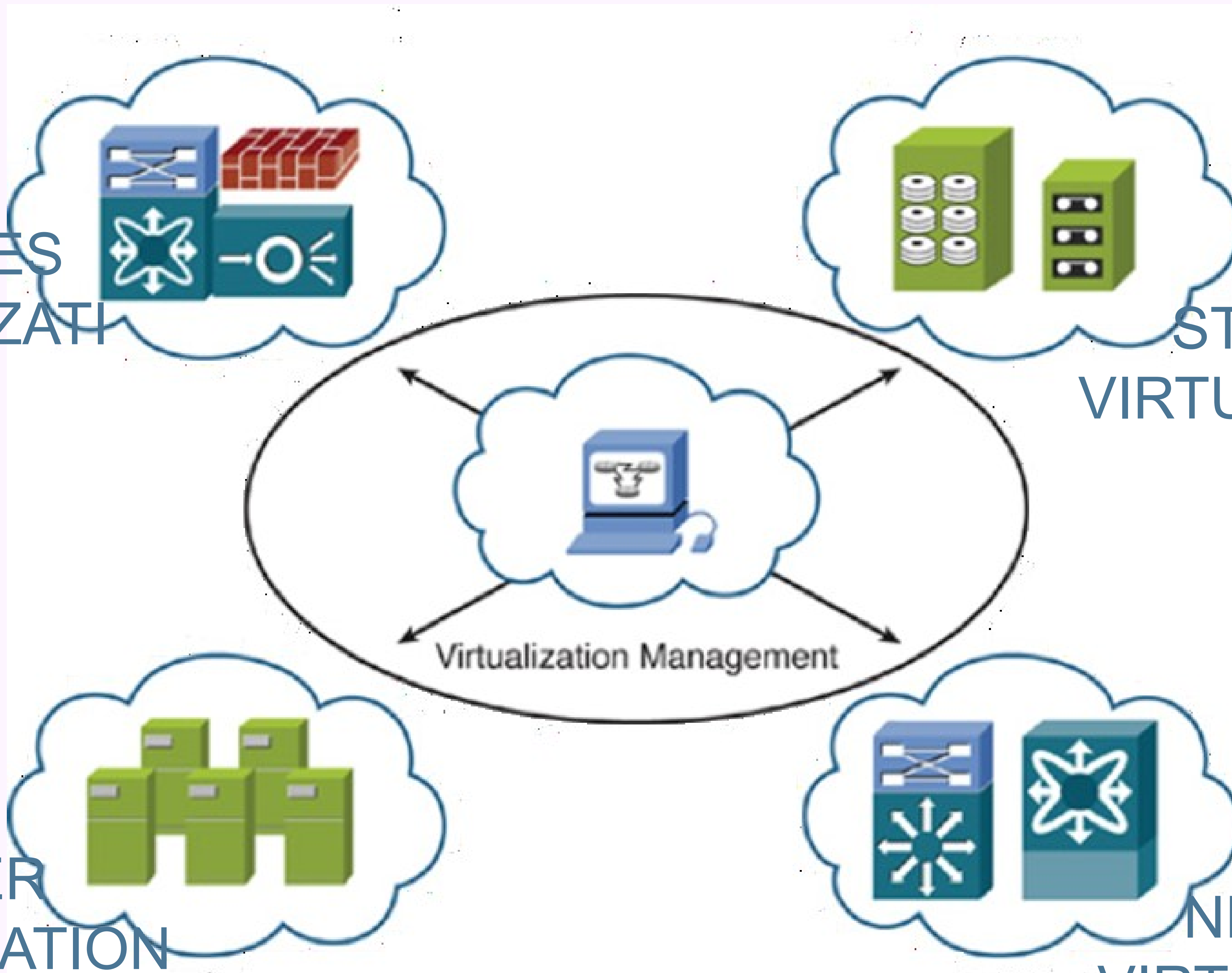
Virtualization

- **Key technology in cloud computing**

- What does it mean?

- Creation of a virtual (rather than actual) version of something, such as an operating system, computing devices (server), storage devices or network devices

Types of Virtualization



SERVICES VIRTUALIZATI ON

STORAGE VIRTUALIZATION

SERVER VIRTUALIZATION

NETWORK VIRTUALIZATION

Group Activity

- (2 minutes) In groups of three decide on the most badass computer that any of you has ever used
  - number of cores
  - frequency
  - memory
  - hard disk