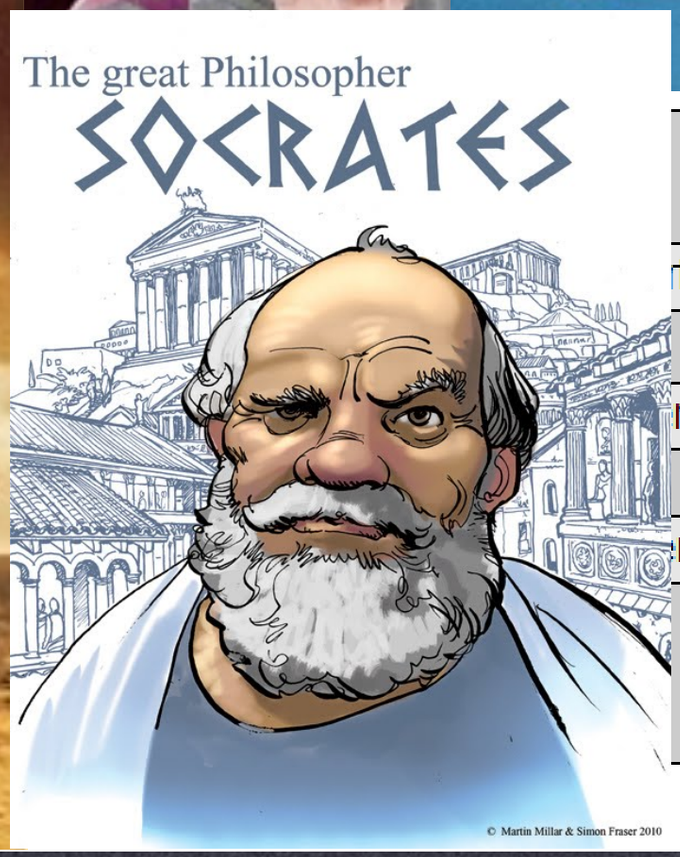
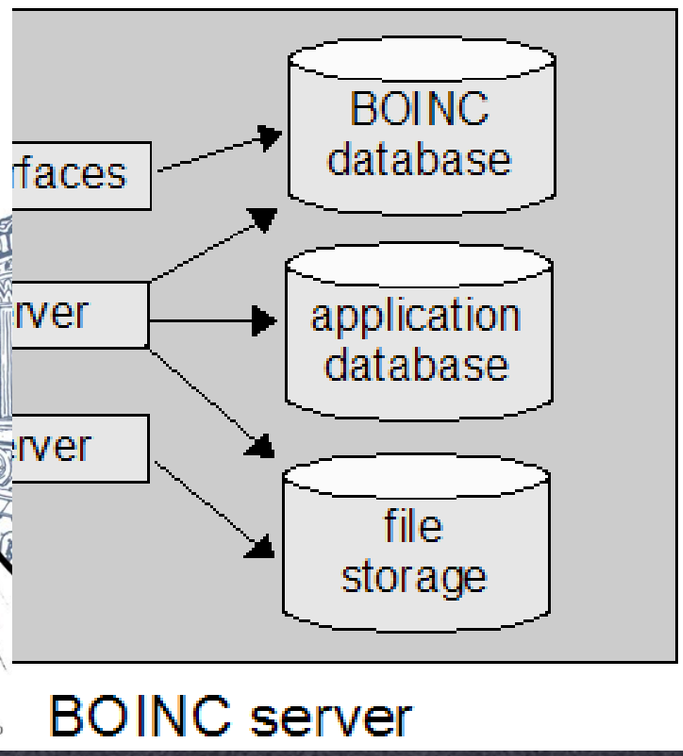




DISTRIBUTED SYSTEMS

SYSTEMS TALK I RPC (CONTINUED)

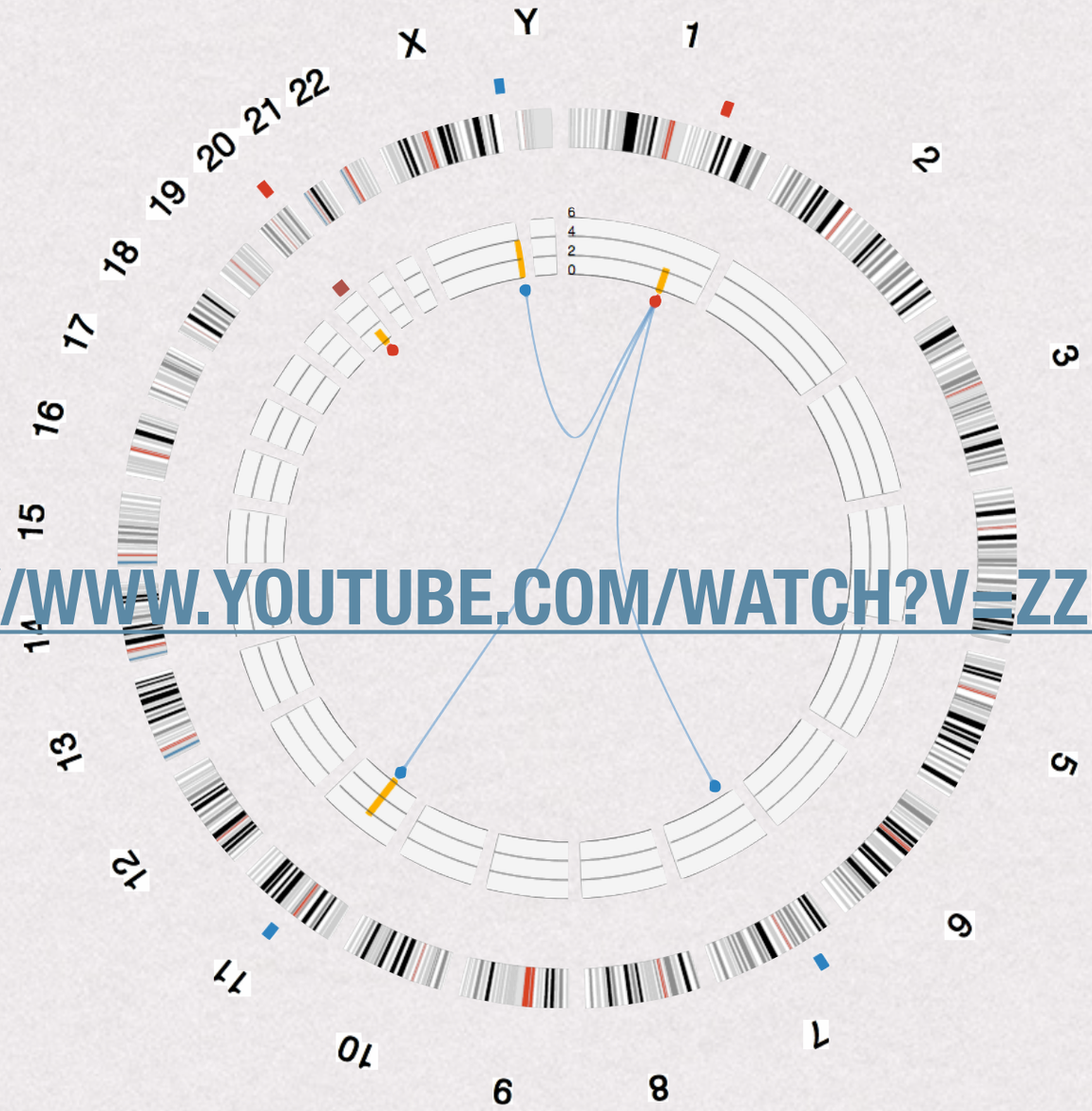
Activity

- What is the most powerful computer that any of you has ever used?
  - number of cores
  - frequency
  - memory
  - hard disk
  - Answer here: https://www.mentimeter.com/app/presentation/alsnfrkirf4y9p3kynim7anx47fjqey5/va1pwcros3wz

SYSTEMS TALK (XKCD 530)

Google Compute Engine

Genome Explorer - powered by Google and Institute for Systems Biology



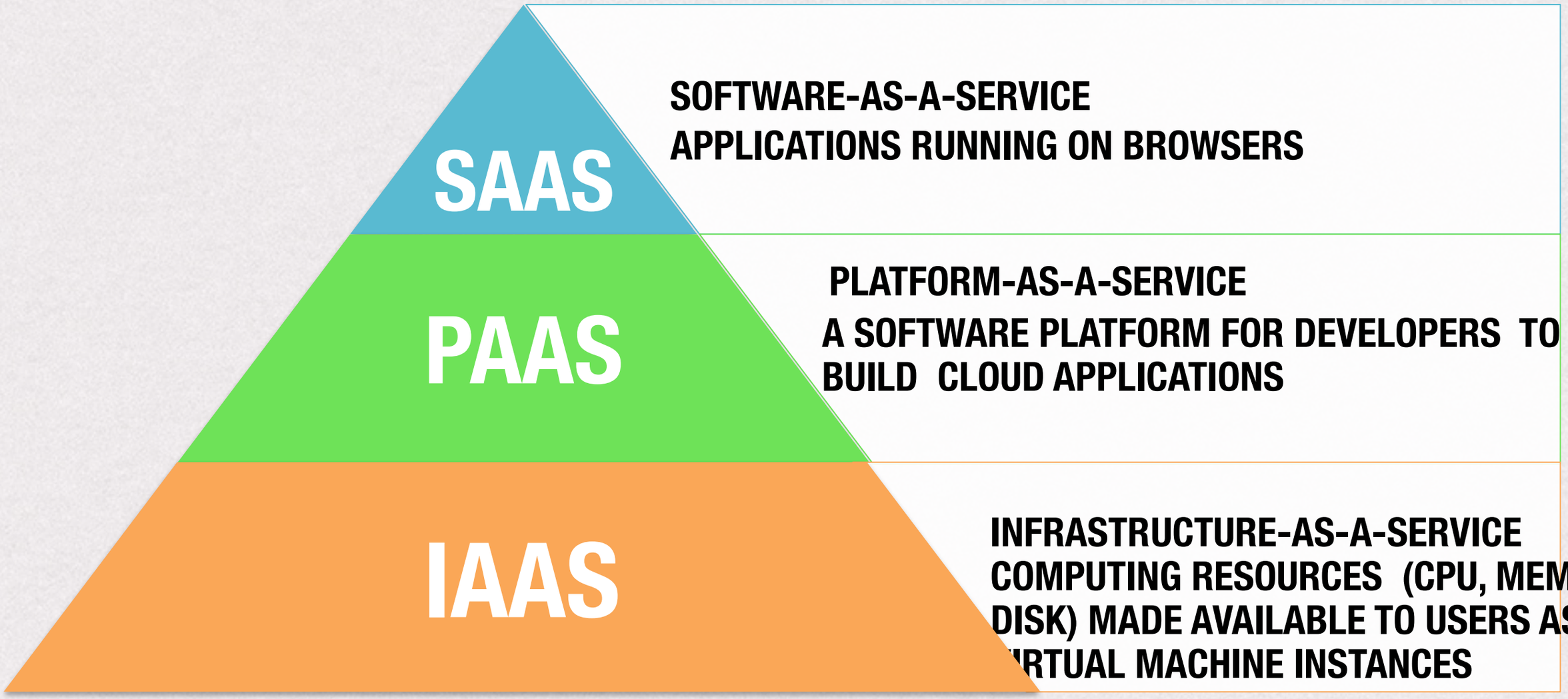
HTTP://WWW.YOUTUBE.COM/WATCH?V=ZZBCVMV-6P4

Google Compute Engine

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

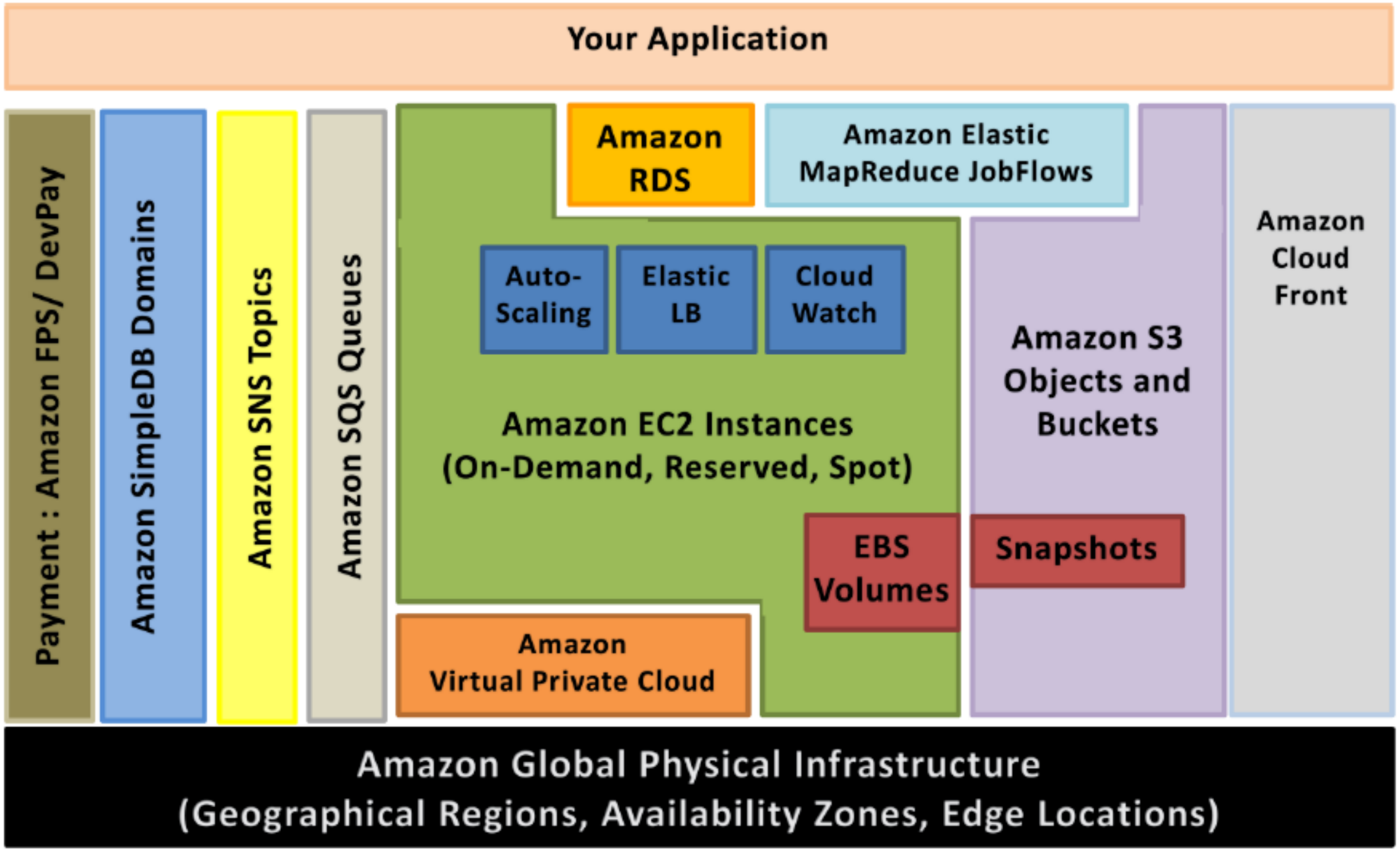
SERVICES VIRTUALIZATION

STORAGE VIRTUALIZATION

SERVER VIRTUALIZATION

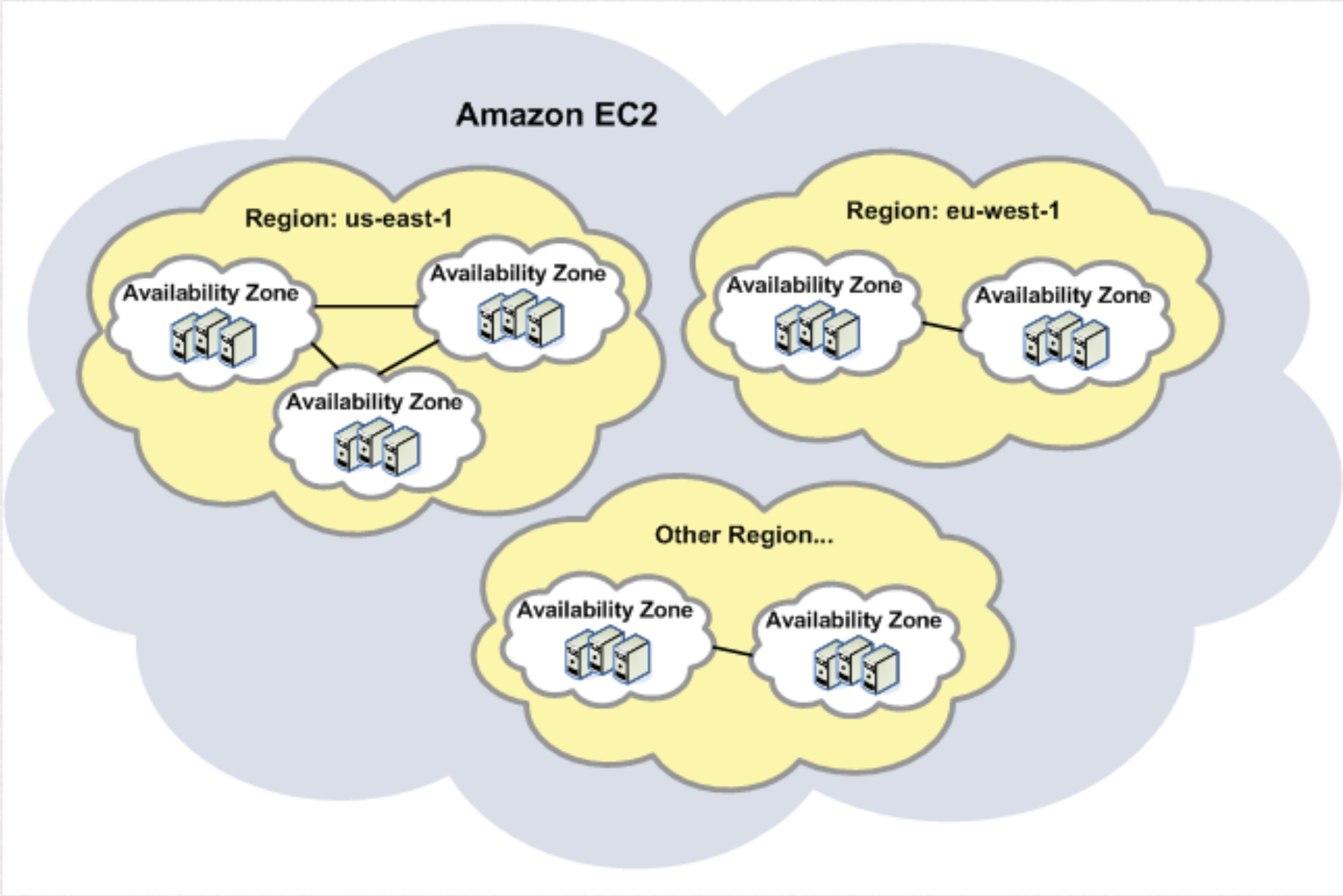
NETWORK VIRTUALIZATION

AWS Ecosystem



J. VARIA, ARCHITECTING FOR THE CLOUD: BEST PRACTICES, AMAZON WEB SERVICES, MAY 2010.

Regions and Availability Zones



Regions and Availability Zones

- Amazon EC2 provides the ability to place instances in multiple locations
- Amazon EC2 locations are composed of Availability Zones and Regions
- Regions are distributed and located in separate geographic areas (US, EU, etc.)
  - You can design your application to be closer to specific customers or to meet legal or other requirements
- Availability Zones are distinct locations within a Region
  - You can protect your applications from the failure of a single location

Amazon's Global Datacenters

AMAZON EC2 IS CURRENTLY AVAILABLE IN EIGHT REGIONS: US EAST (NORTHERN VIRGINIA), US WEST (OREGON), US WEST (NORTHERN CALIFORNIA), EU (IRELAND), ASIA PACIFIC (SINGAPORE), ASIA PACIFIC (TOKYO), SOUTH AMERICA (SAO PAULO), AND AWS GOVCLOUD

Remote Procedure Call

- 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…)

Group Activity

- Java's garbage collection can (simplistically) be implemented using reference counting
- How would an RMI distributed garbage collection algorithm work?
  - what different kinds of objects are there in RMI?
  - how would you deal with them?
  - where would your DGC be (on the client? on the server?)
  - Please do not use any internet-capable devices

Group Activity

- When the client creates (unmarshalls) a remote reference, it calls dirty() on the server-side DGC. After the client has finished with the remote reference, it calls the corresponding clean() method.
- A reference to a remote object is leased for a time by the client holding the reference.
- Lease starts when the dirty() call is received.
- The client must renew the leases by making additional dirty() calls on the remote references it holds before such leases expire.
- If the client does not renew the lease before it expires, the distributed garbage collector assumes that the remote object is no longer referenced by that client.

Last week

- RPC
- Reducing remote operation latency
  - Latency hiding
  - Latency reduction

Latency Hiding – Readings

- Futures and promises paper:
  - B. Liskov and L. Shrira. Promises: Linguistic support for efficient asynchronous procedure calls in distributed systems. In Proceedings of the ACM SIGPLAN '88 Conference on Programming Languages Design and Implementation. ACM, June 1988. 63

Latency Hiding

- Two camps:
  - system level - multi-threading OS, language support for general threading
  - language-level - futures and promises; superior?

Recall …

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

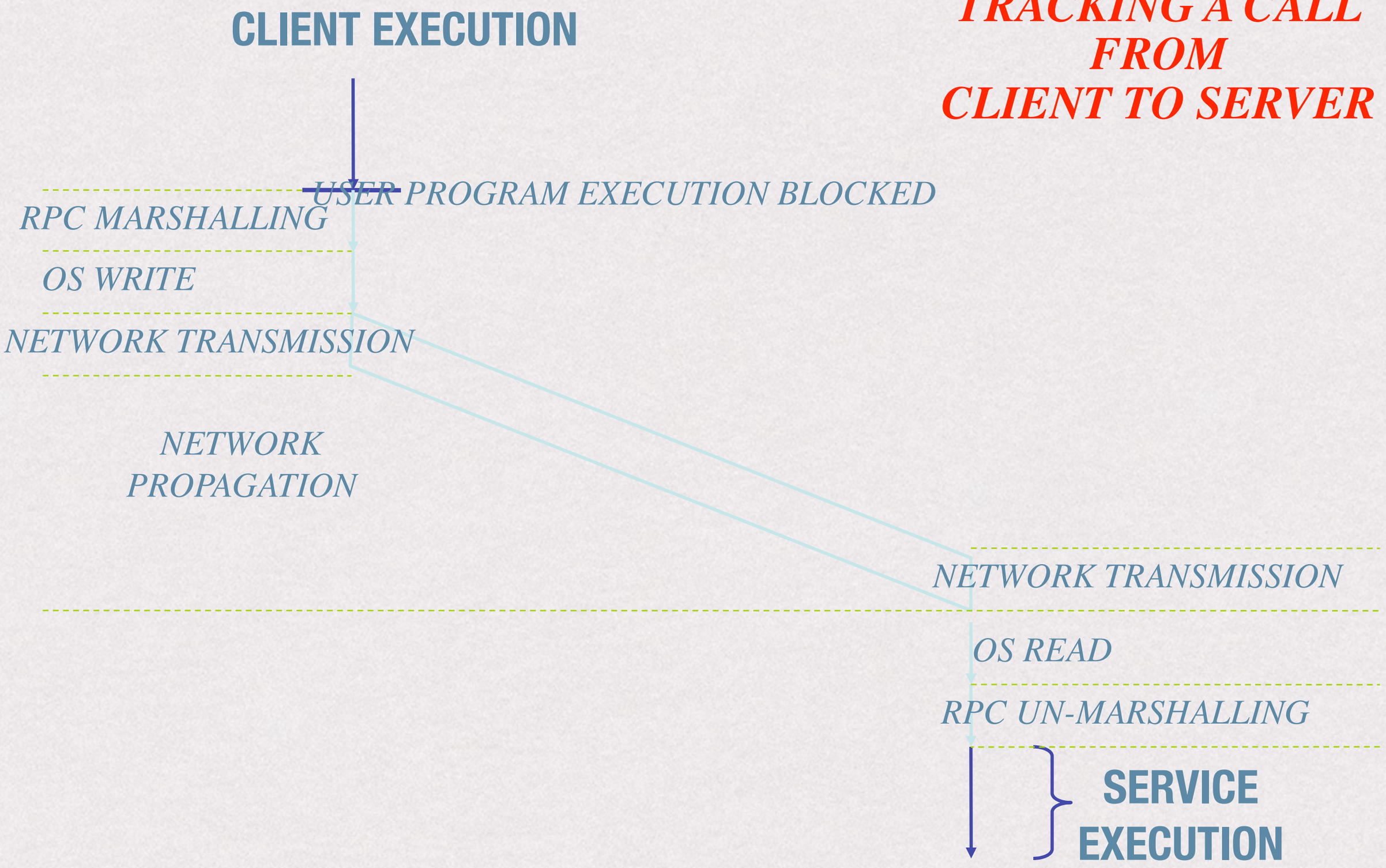
Recall … Average Latency

Total time taken for client divided by the number of operations

vs

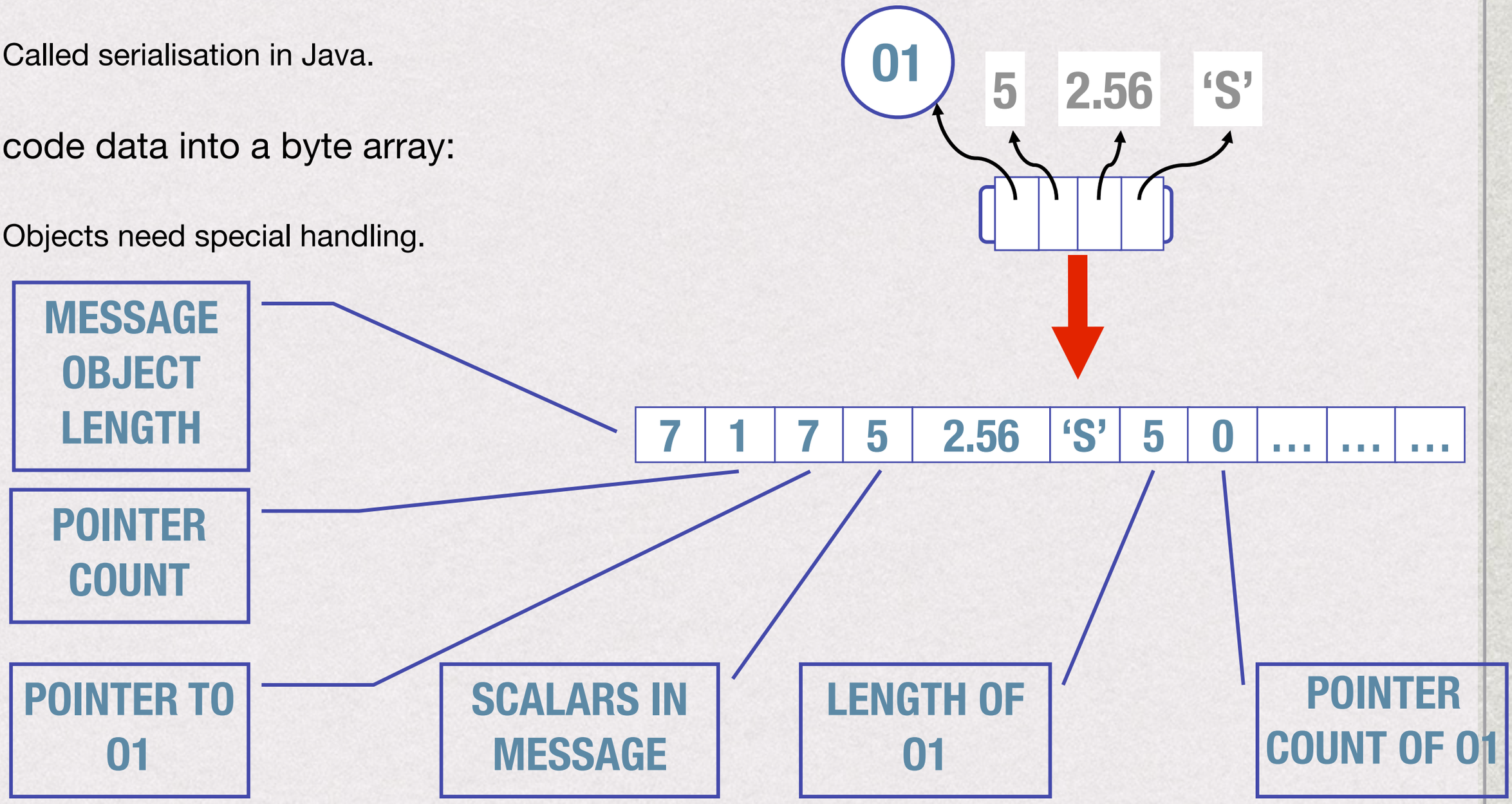
Average time taken for each operation individually

TRACKING A CALL FROM CLIENT TO SERVER



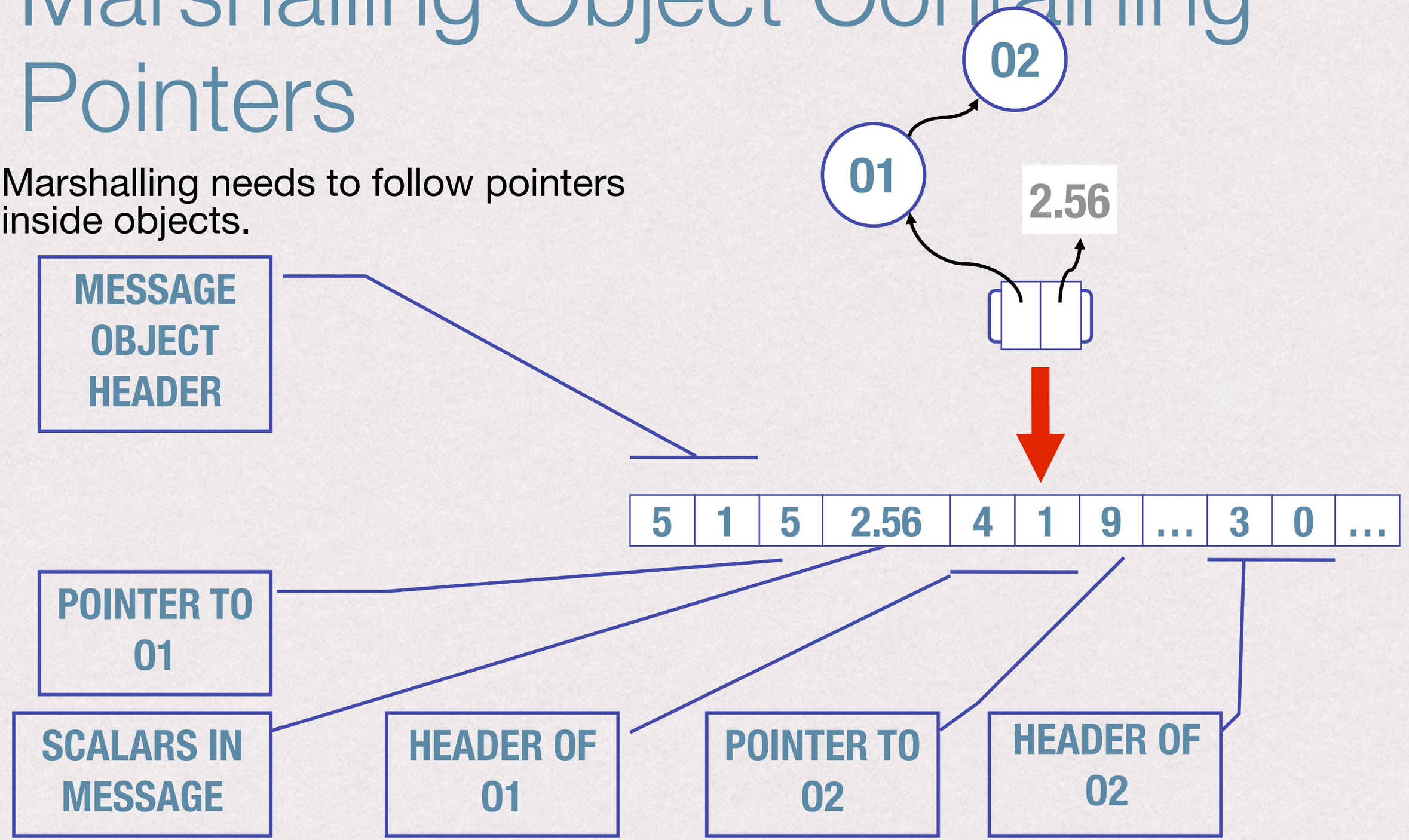
Marshalling

- The process of flattening data to send a copy of it over a network:
  - Called serialisation in Java.
- Encode data into a byte array:
  - Objects need special handling.

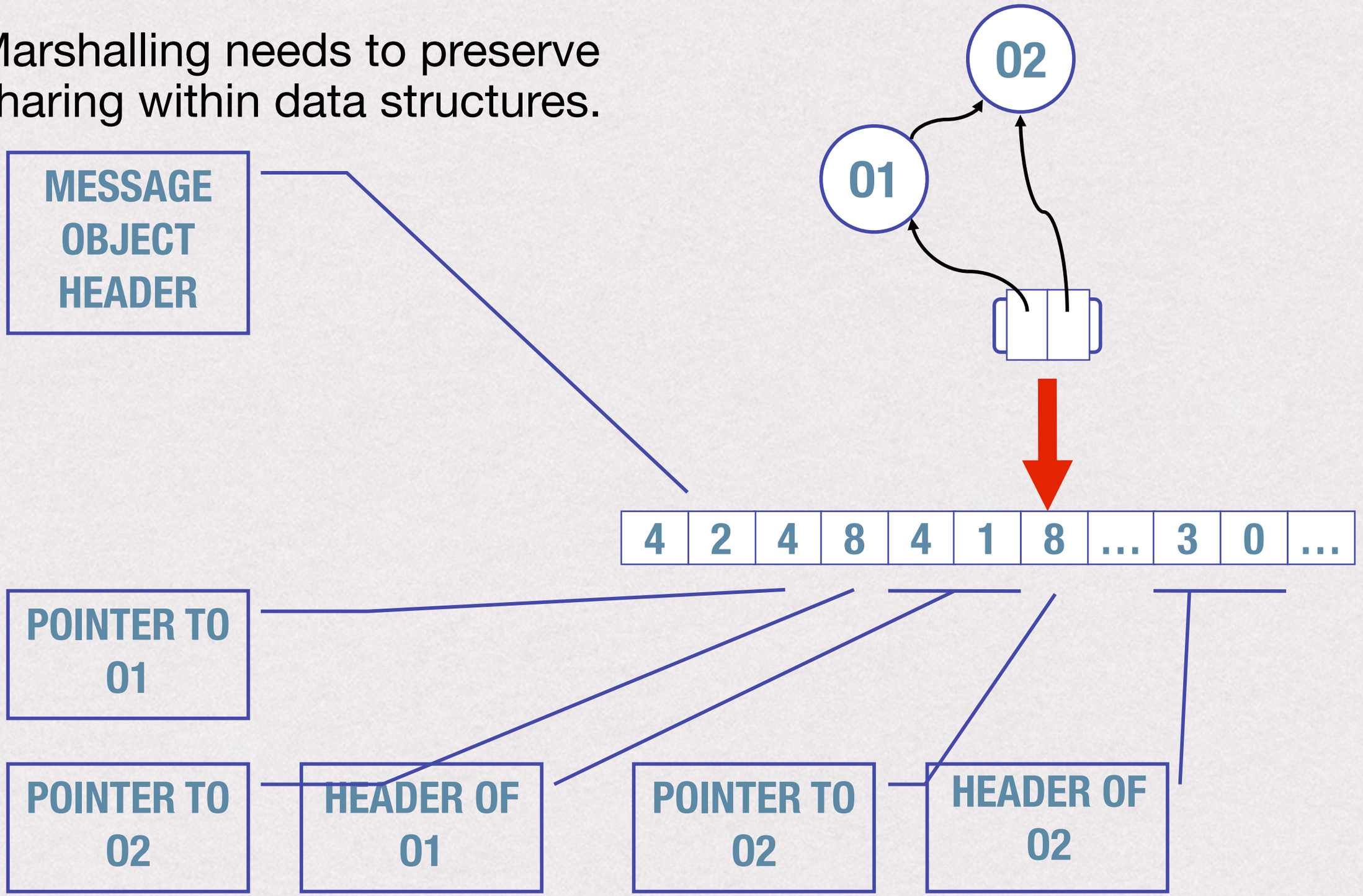


Marshalling Object Containing Pointers

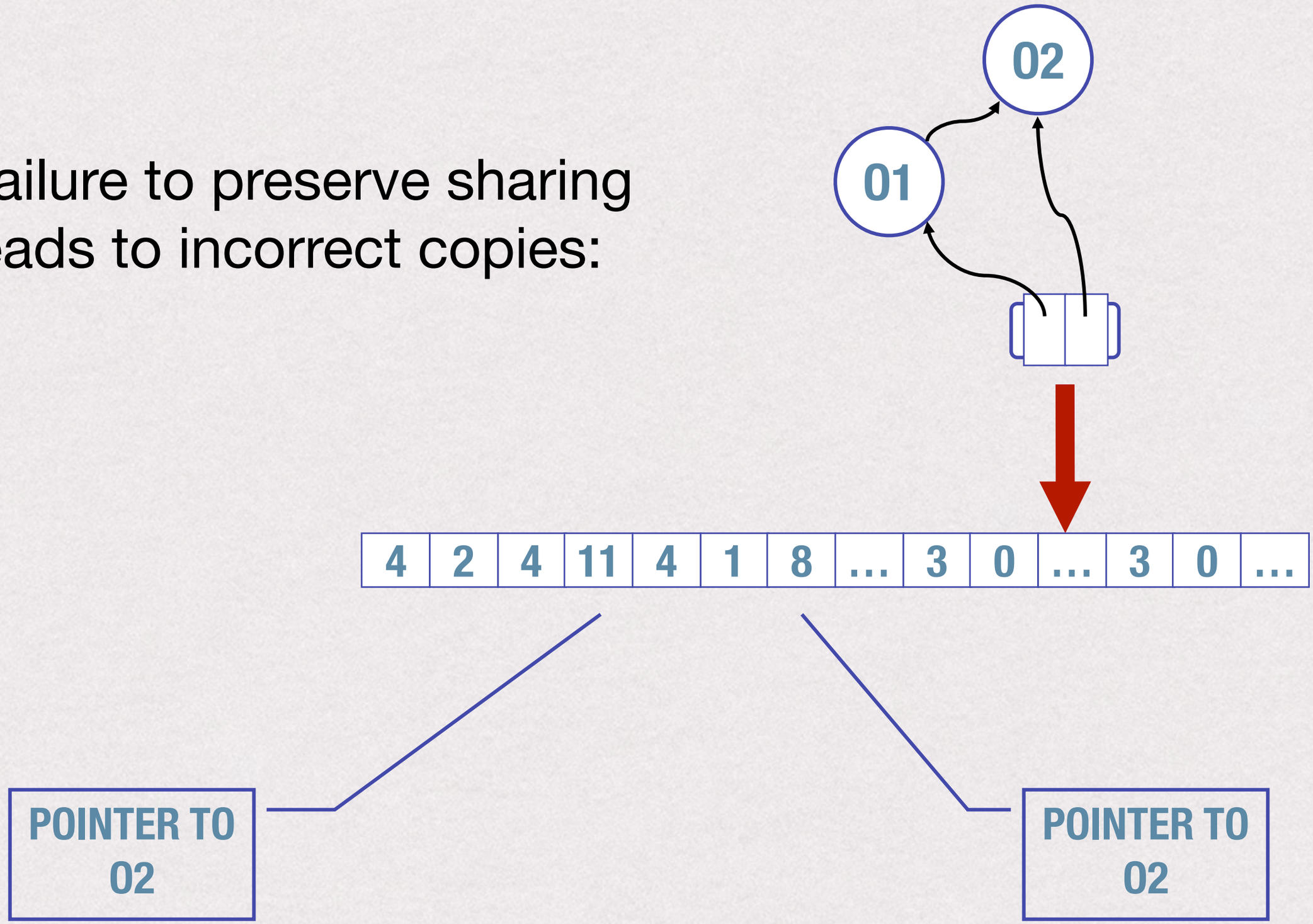
- Marshalling needs to follow pointers inside objects.



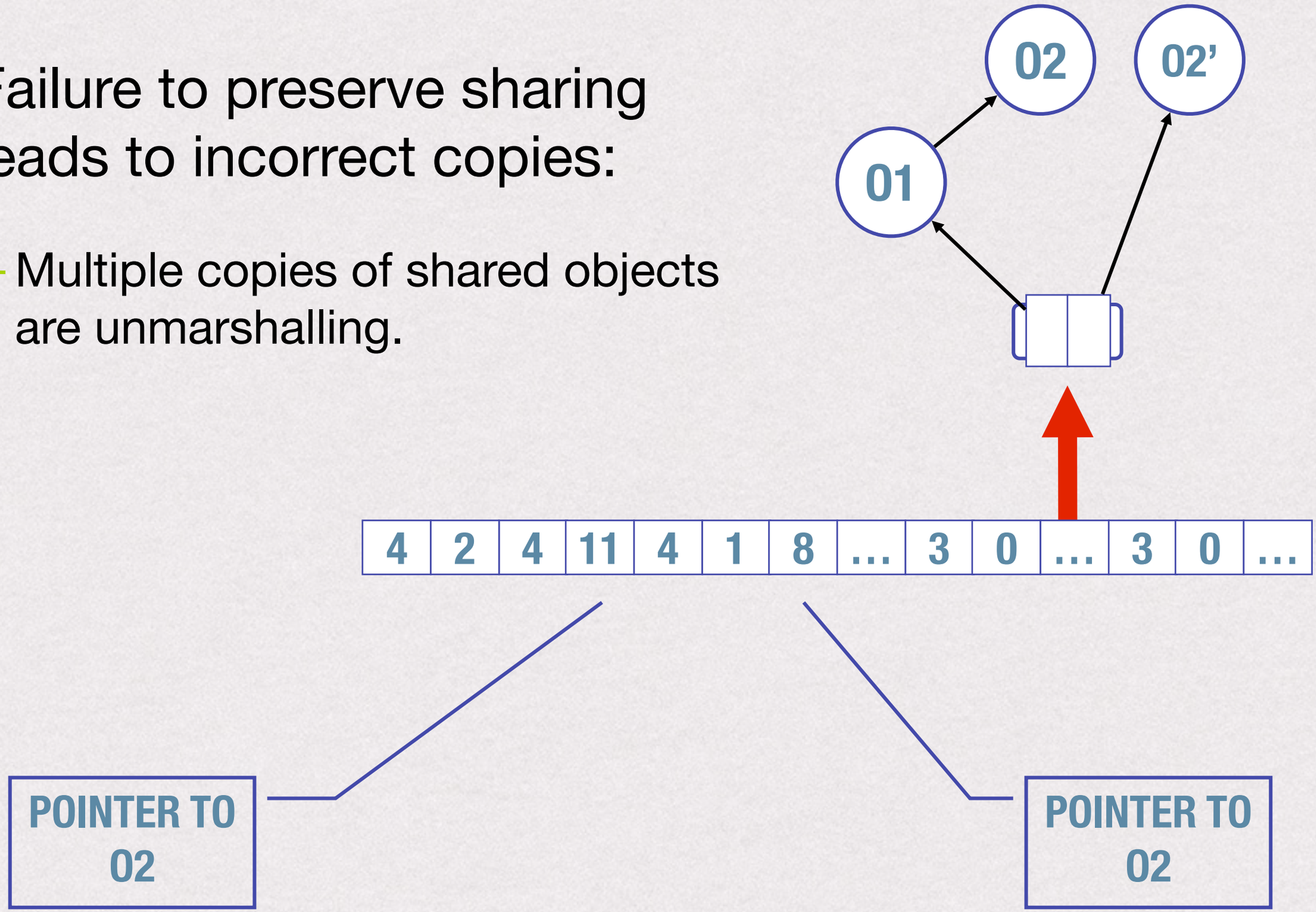
- Marshalling needs to preserve sharing within data structures.



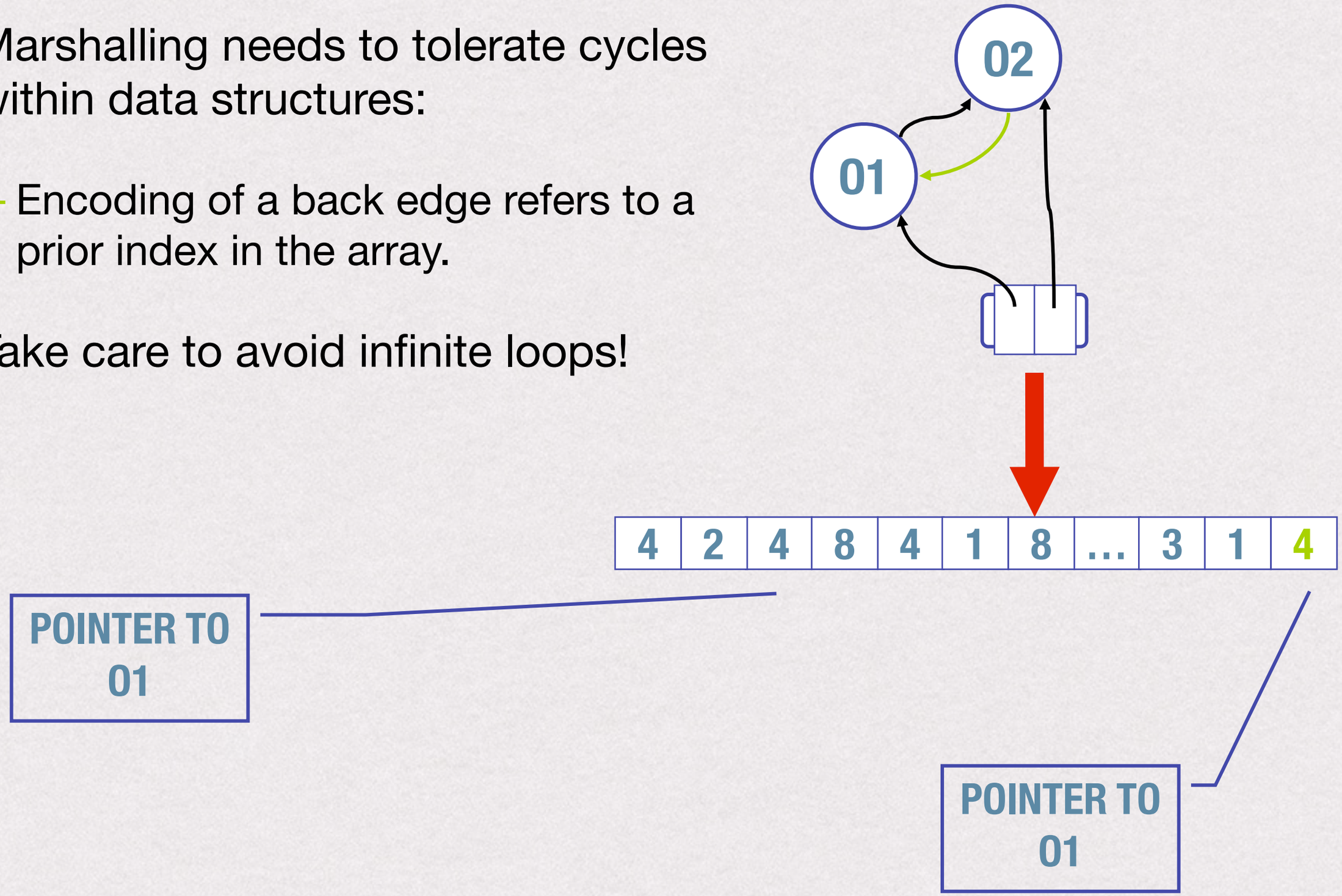
- Failure to preserve sharing leads to incorrect copies:



- Failure to preserve sharing leads to incorrect copies:
  - Multiple copies of shared objects are unmarshalling.



- Marshalling needs to tolerate cycles within data structures:
  - Encoding of a back edge refers to a prior index in the array.
- Take care to avoid infinite loops!



Latency Reduction Issues

- Network propagation delay is usually dominant
  - Other terms reducible by increasing network bandwidth (transmission delay) or processing power (all else).
  - Lower bound on propagation delay due to speed of light .
- Latency reduction aims to reduce the mean count of propagation delays per remote call:
  - For a single RPC, the number of propagation delays per call is 2/1, i.e. two – one for the call, one for the result.
  - Alternative terminology is to talk in terms of “round trips” a term referring to a pair of propagation delays.

Focus

- Dependencies constraints
  - Classify different types of dependencies
- Abstractions to reduce propagation delays

Latency Reduction with Parallel RPC

1

```
pcall {
    res1 = O1.m1(paramsA …)
    res2 = O2.m2(paramsB …)
    res3 = O3.m3(paramsC …)
}
```

- All three (or however many) calls are sent off together, in any order, then the client blocks until the results of all calls have arrived back.
- *What is the overall latency for the calls in a parallel RPC?*
- *What is the average latency for the calls in a parallel RPC?*
- *What dependencies are permitted between the calls in a parallel RPC?*

Latency Reduction with One-way RPC

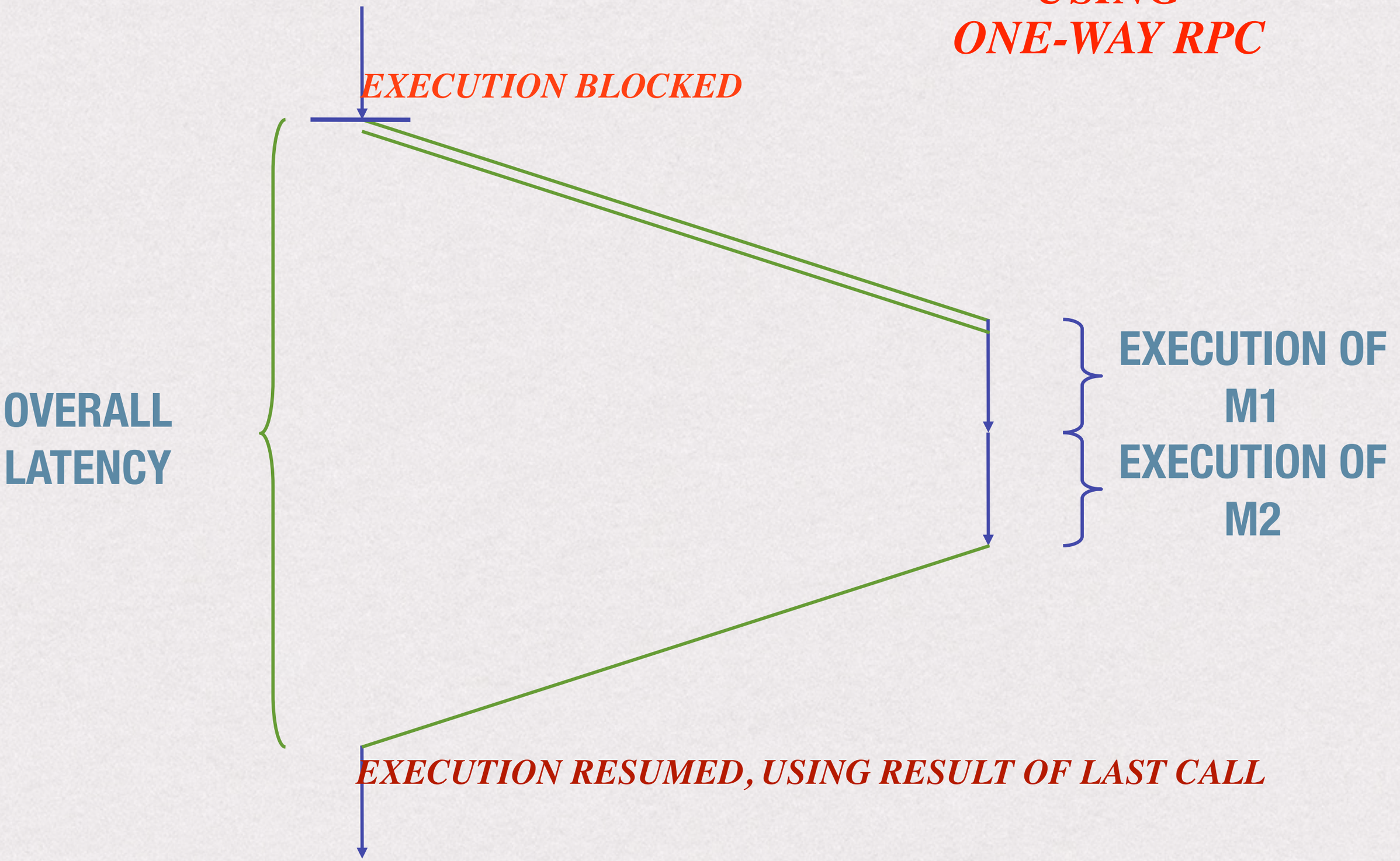
2

- The ***One-way RPC*** idea is:

```
O1.m1(paramsA …)
res2 = O1.m2(paramsB …)
```

  - All except the last of this group of calls are one-way RPCs – they produce no result.

LATENCY REDUCTION USING ONE-WAY RPC



One-way RPC

- The ***One-way RPC*** idea is:

```
O1.m1(paramsA …)
res2 = O1.m2(paramsB …)
```

  - All except the last of this group of calls are one-way RPCs – they produce no result.

- *What is the overall latency for a series of calls using one-way RPC?*
- *What is the mean latency for a series of calls using one-way RPC?*
- *What can we say about success or failure of one-way RPC calls?*
- *What can be said about the server used to process the calls in such a group?*
- *What dependencies are permitted between the calls using one-way RPC?*
- *There is a variant of one-way RPC call maybe RPC, i.e. the call may be executed, but there is no way to find out whether it did*

Latency Reduction with Futures and Promises

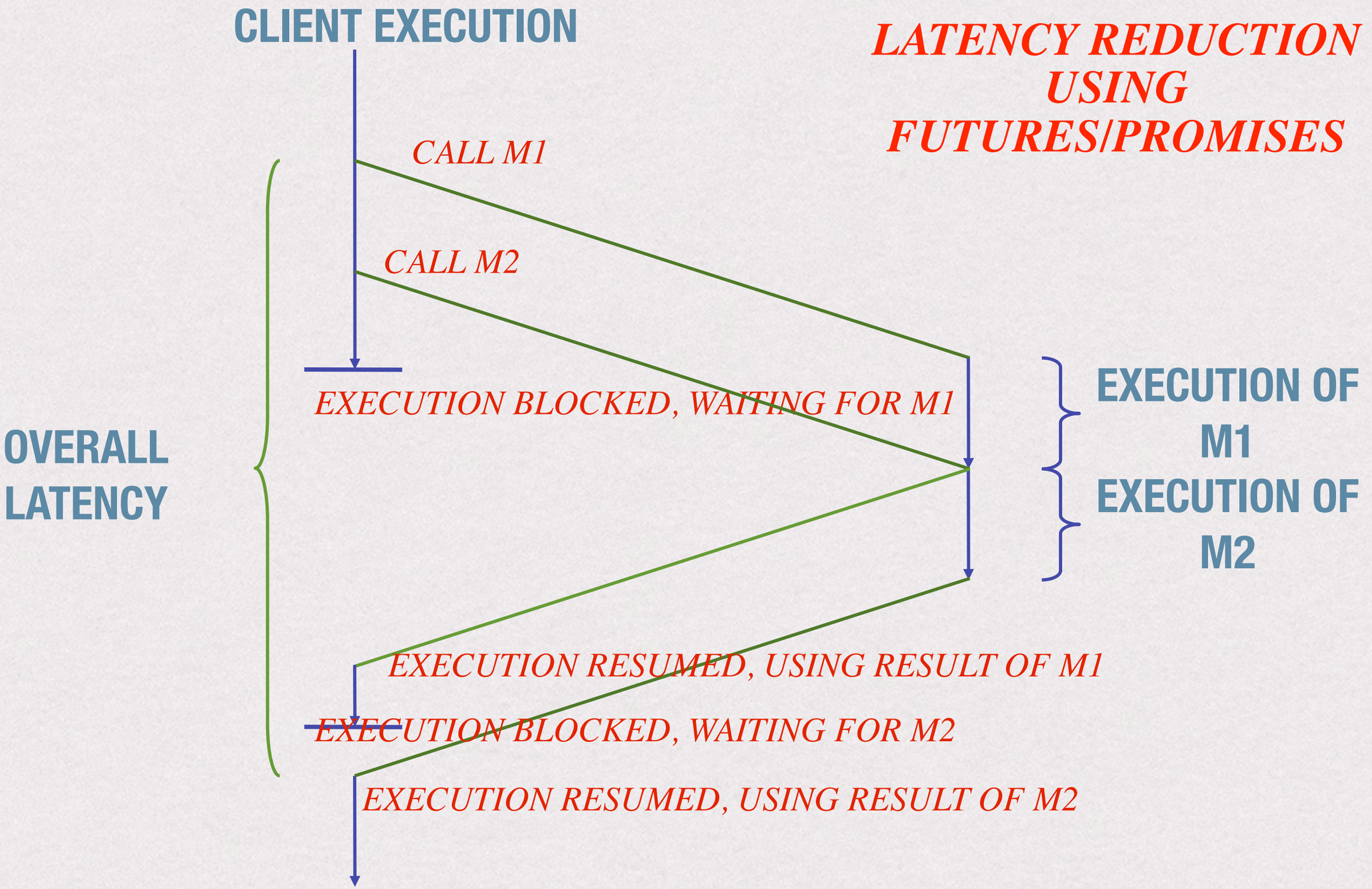
3

- An example of the Future/Promises idea is:

```
res1 = O1.m1(paramsA …)
… computation …
res2 = O1.m2(paramsB …)
… computation …
… computation using res1 …
… computation using res2 …
res3 = O2.m3(paramsC …)
```

- Much more flexible than Parallel RPC!

LATENCY REDUCTION USING FUTURES/PROMISES



Futures and Promises

3

- An example of the Future/Promises idea is:

```
res1 = O1.m1(paramsA …)
… computation …
res2 = O1.m2(paramsB …)
… computation …
… computation using res1 …
… computation using res2 …
res3 = O2.m3(paramsC …)
```

- What is the best case for total latency of m1 and m2 considered together?
- What is the best case for mean latency of m1 and m2 considered together?
- What can we say about the dependencies between m1 and m2?
- What can we say about the dependencies between m2 and m3, assuming the objects called are on different servers?

Recall …

- Latency reduction strategy:
  - It became increasingly difficult to analyse as the mechanisms became more complex, more application dependent
  - However, as mechanisms required less blocking they tended to achieve better latency reduction.
- In this part of the lecture, we will explore this tendency further.

Use of Results and Blocking

- Consider the following:

```
res1 = O.m1(…)
res2 = O.m2(…, res1, …)
```

- With explicit claims, this must be re-written:

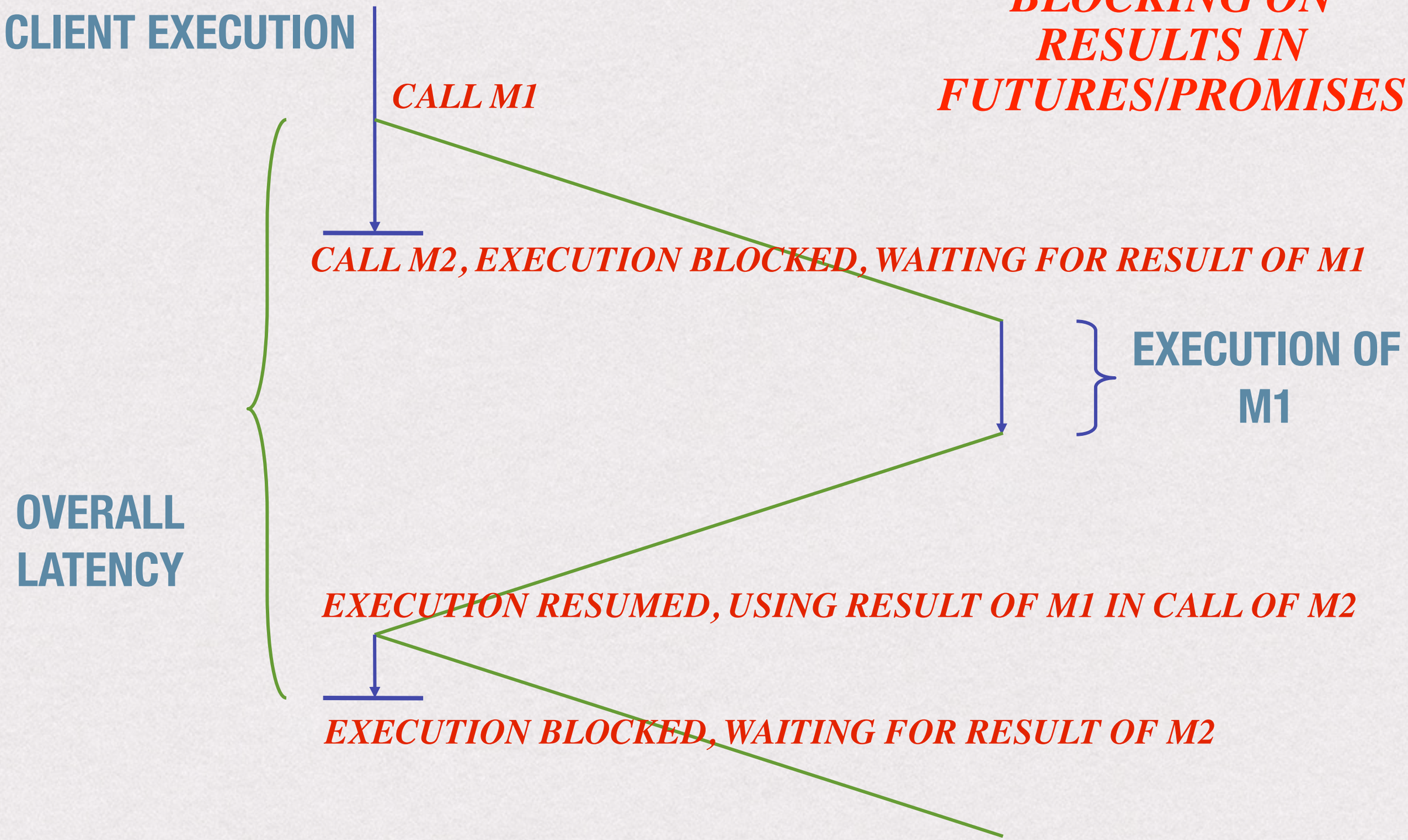
```
res1 = O.m1(…)
res2 = O.m2(…, claim(res1), …)
```

- With implicit claims, also known as Wait by Necessity, the code can be used as is.

- In either case, the claim operation (whether implicit or explicit) may block the client:

- it will block if the result has not arrived, and until it does arrive.

BLOCKING ON RESULTS IN FUTURES/PROMISES



Can we do better than this?

Latency Reduction with Batched Futures

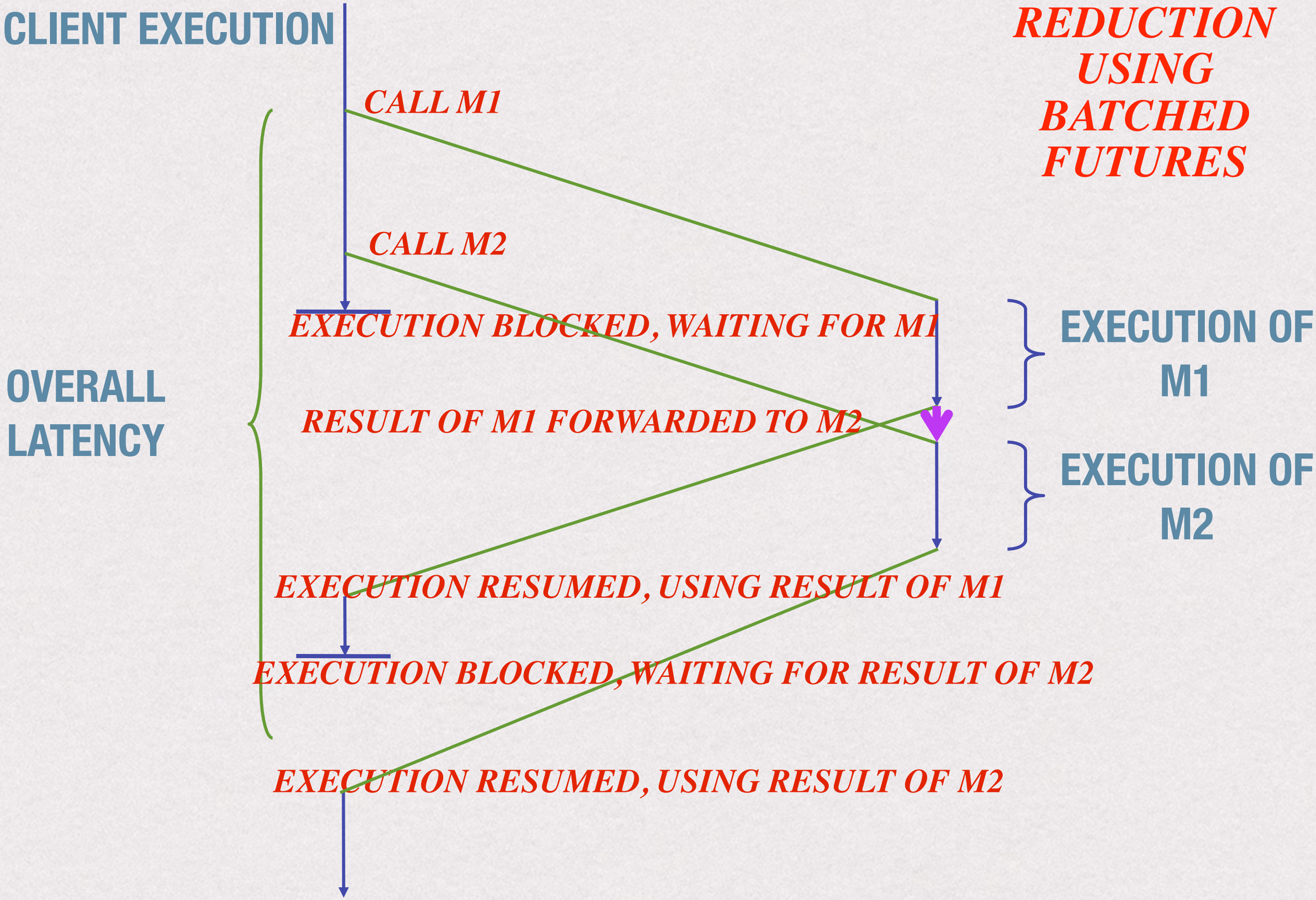
4

- An example of the Batched Futures idea is:

```
res1 = O1.m1(paramsA …)
… computation …
res2 = O1.m2(…, res1, …)
```

- The result of the first call is able to be used as a parameter of the second call, without the possibility of blocking.

LATENCY REDUCTION USING BATCHED FUTURES



Batched Futures

```
res1 = O1.m1(paramsA …)
… computation …
res2 = O1.m2(…, res1, …)
```

- What is the longest path (i.e critical path) length in the network propagation graph?
- What is the best case for total latency of m1 and m2 considered together?
- What is the best case for mean latency of m1 and m2 considered together?
- What can we say about the dependencies between m1 and m2?
- Using batched futures, must m1 and m2 be calls to object on the same server?

Latency Reduction with Responsibilities

5

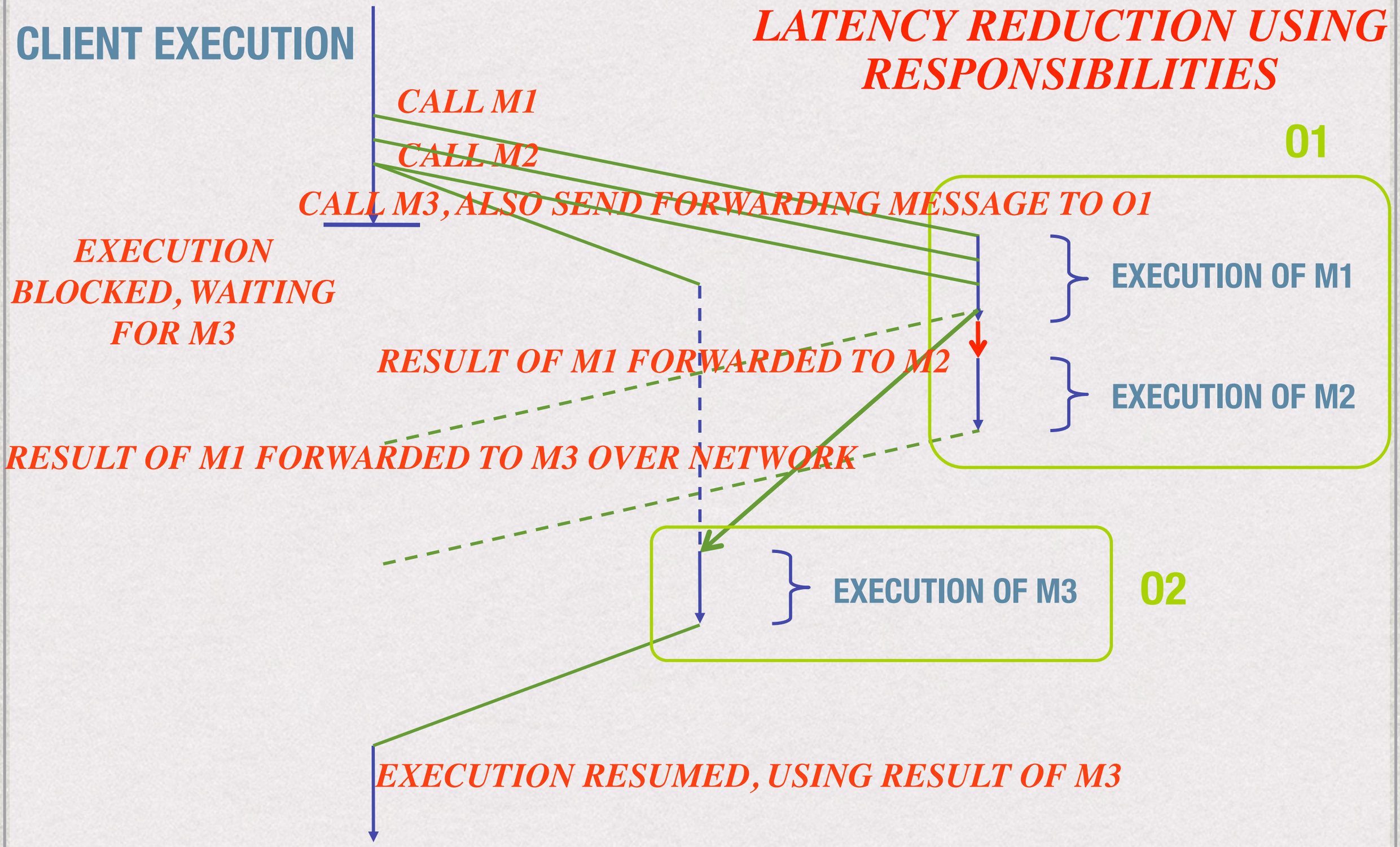
- An example of the Responsibilities idea is:

```
res1 = O1.m1(paramsA …)
res2 = O1.m2(…, res1, …)
res3 = O2.m3(…, res1, …)
x = res3 + 5
```

- With Batched Futures, the third call will block the client until res1 arrives, since the call is being sent to a different server (object).

- With Responsibilities, the third call is also non-blocking.

LATENCY REDUCTION USING RESPONSIBILITIES



Responsibilities

```
res1 = O1.m1(paramsA …)
res2 = O1.m2(…, res1, …)
res3 = O2.m3(…, res1, …)
x = res3 + 5
```

- What is the longest path length through the network propagation graph?
- What is the best case for total latency of m1, m2 and m3 considered together?
- What would it be for batched futures?
- What is the best case for mean latency of m1, m2 and m3 considered together?
- What would it be for batched futures?

Analyse That!

- One factor in requiring less blocking is the kinds of dependencies between calls that can be traversed without blocking:
  - Result dependent < Order dependent < Independent
- Another factor is whether dependencies between calls to different servers can be traversed without blocking:
  - Multi server < Single server

Types of Dependencies

- Calls m1 and m2:
- ***Independent*** – m1 and m2 can execute in any order and the result of either is not used in the other.
- ***Order Dependent*** – m2 must begin execution after the end of the execution of m1, but m2 does not use the result of m1.
- ***Result Dependent*** – m2 takes as parameters one or more results of m1, and hence must begin execution after the end of the execution of m1:
- Res1 = O.m1(…)
  Res2 = O.m2(…, res1, …)

Types of Dependencies

- ***Functionally Dependent*** – m2 takes as parameters one or more non-identity functions of one or more results of m1,
- must begin execution after the end of the execution of m1:
- Res1 = O.m1(…)
  Res2 = O.m2(…, res1 + 3 , …)