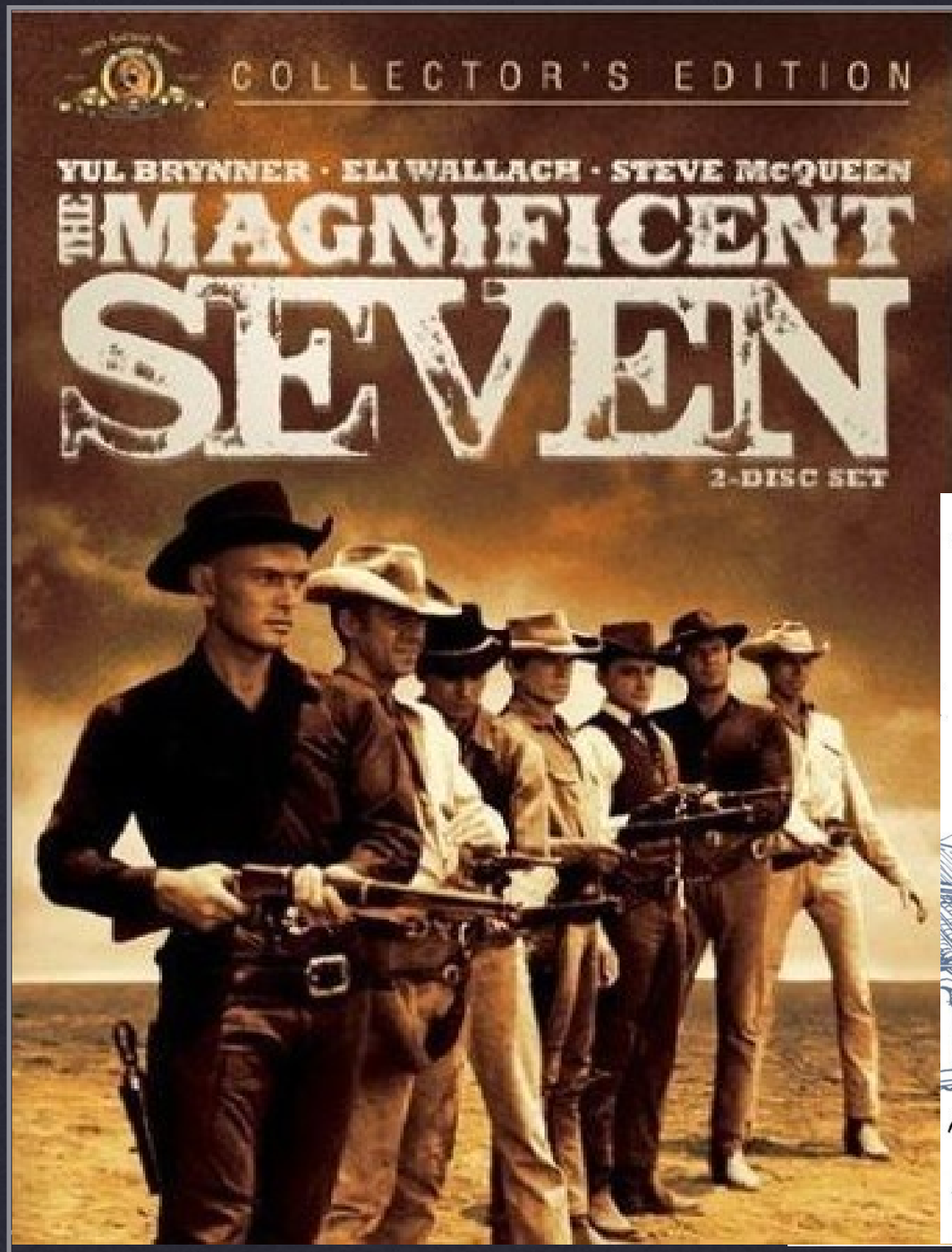
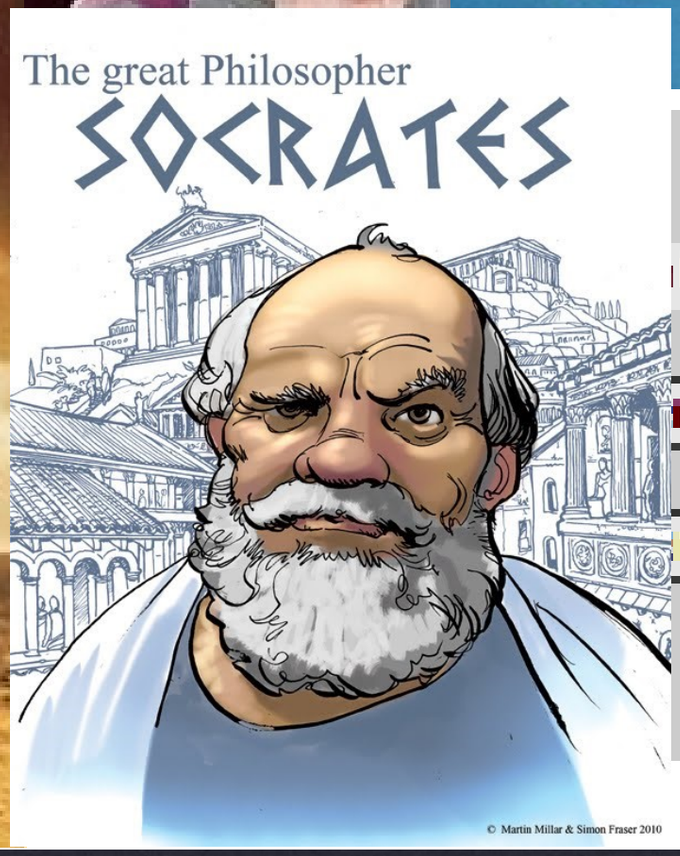
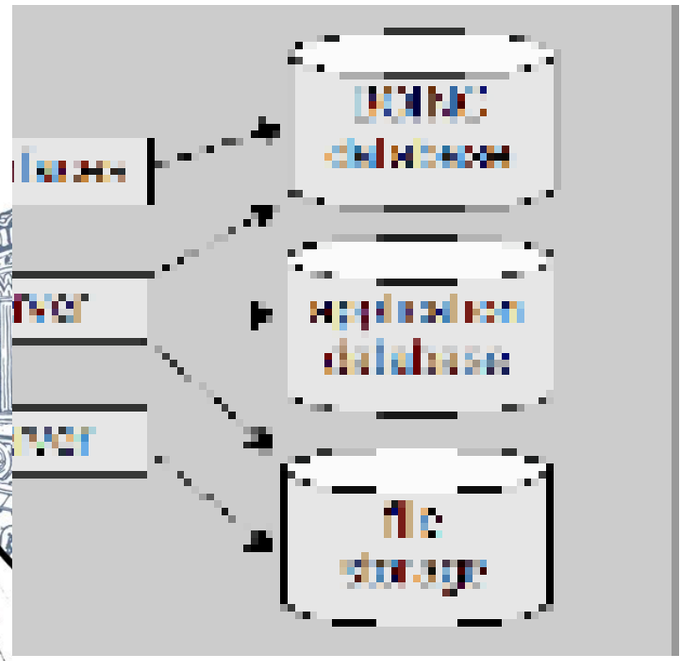






distributed systems

RPC (CONTINUED)

Last week

- Cloud Computing
- 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

Last week

- Cloud Computing
- 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

Barbara Liskov

- One of the first women in US to be awarded PhD in CS
- Research focuses on byzantine fault tolerance & distributed computing
- 2004: John von Neumann medal for "fundamental contributions to programming languages, programming methodology, and distributed systems”
- 2008: Turing Award for her work in the design of programming languages and software methodology that led to the development of object-oriented programming

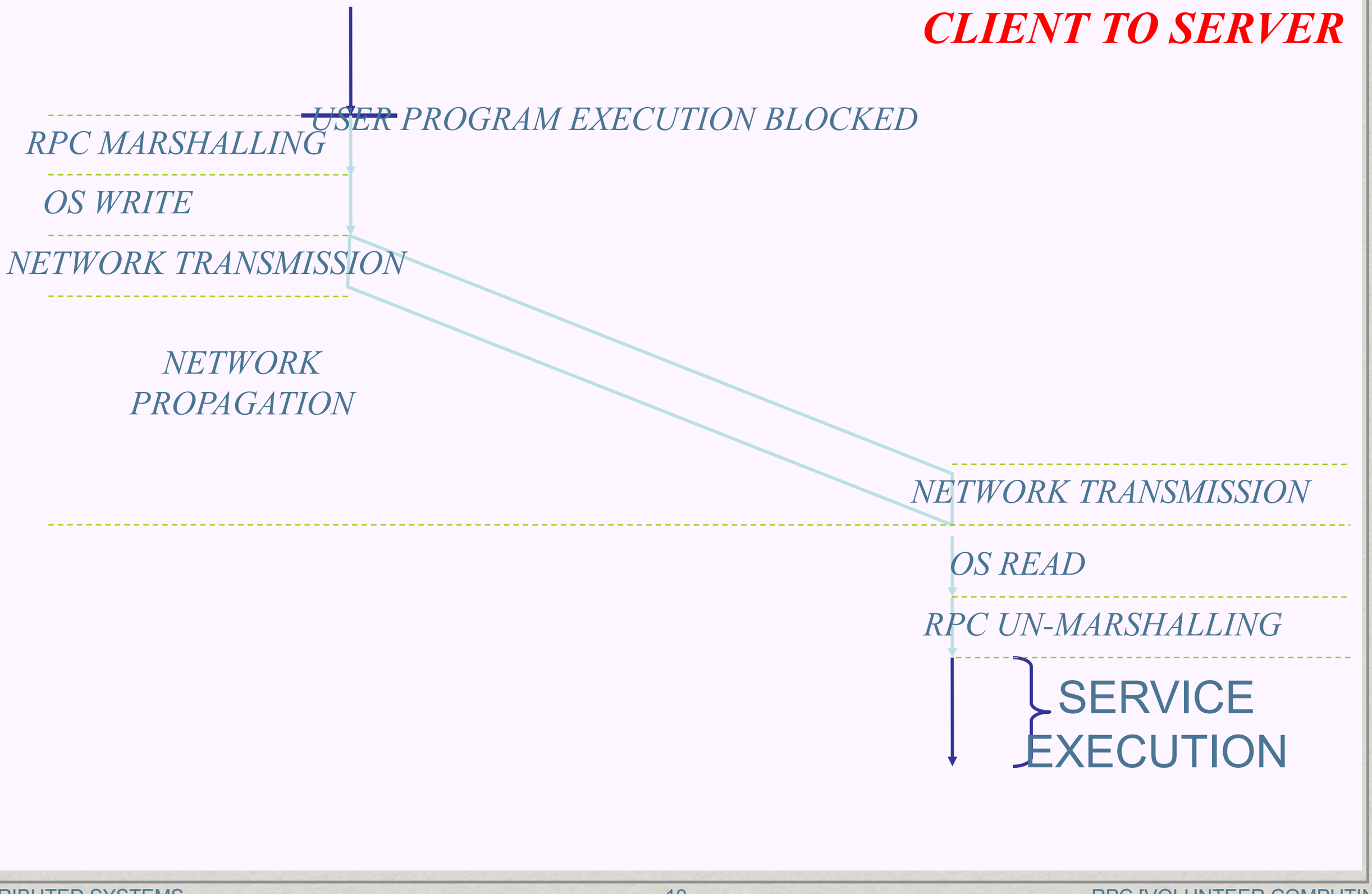
Latency Hiding

- 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.

TRACKING A CALL FROM CLIENT TO SERVER



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 …)
}
```

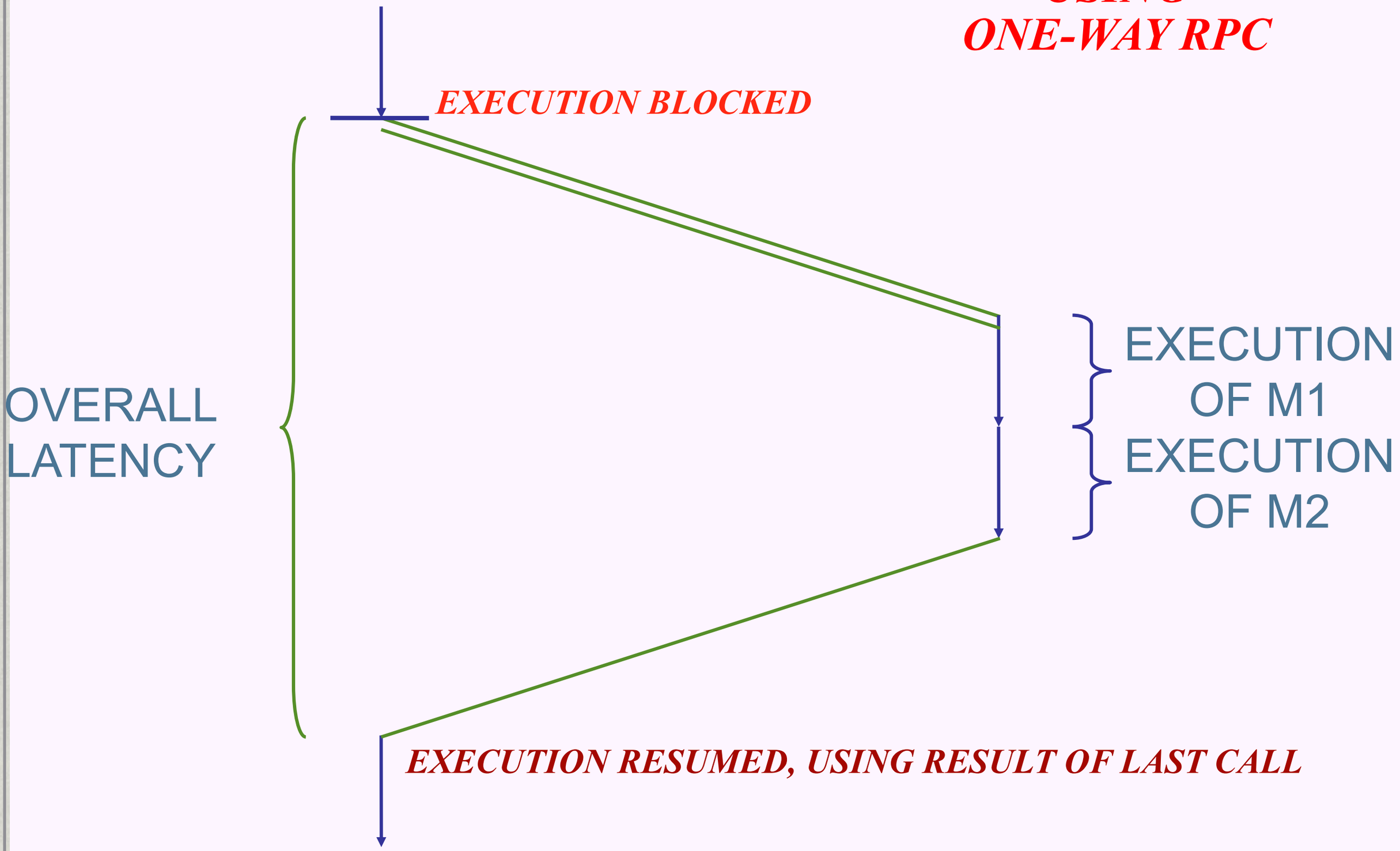
- All three (or however many) calls are sent off together, in any order, then the client blocks until the results of all calls have arrived back.
- *What is the overall latency for the calls in a parallel RPC?*
- *What is the average latency for the calls in a parallel RPC?*
- *What dependencies are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

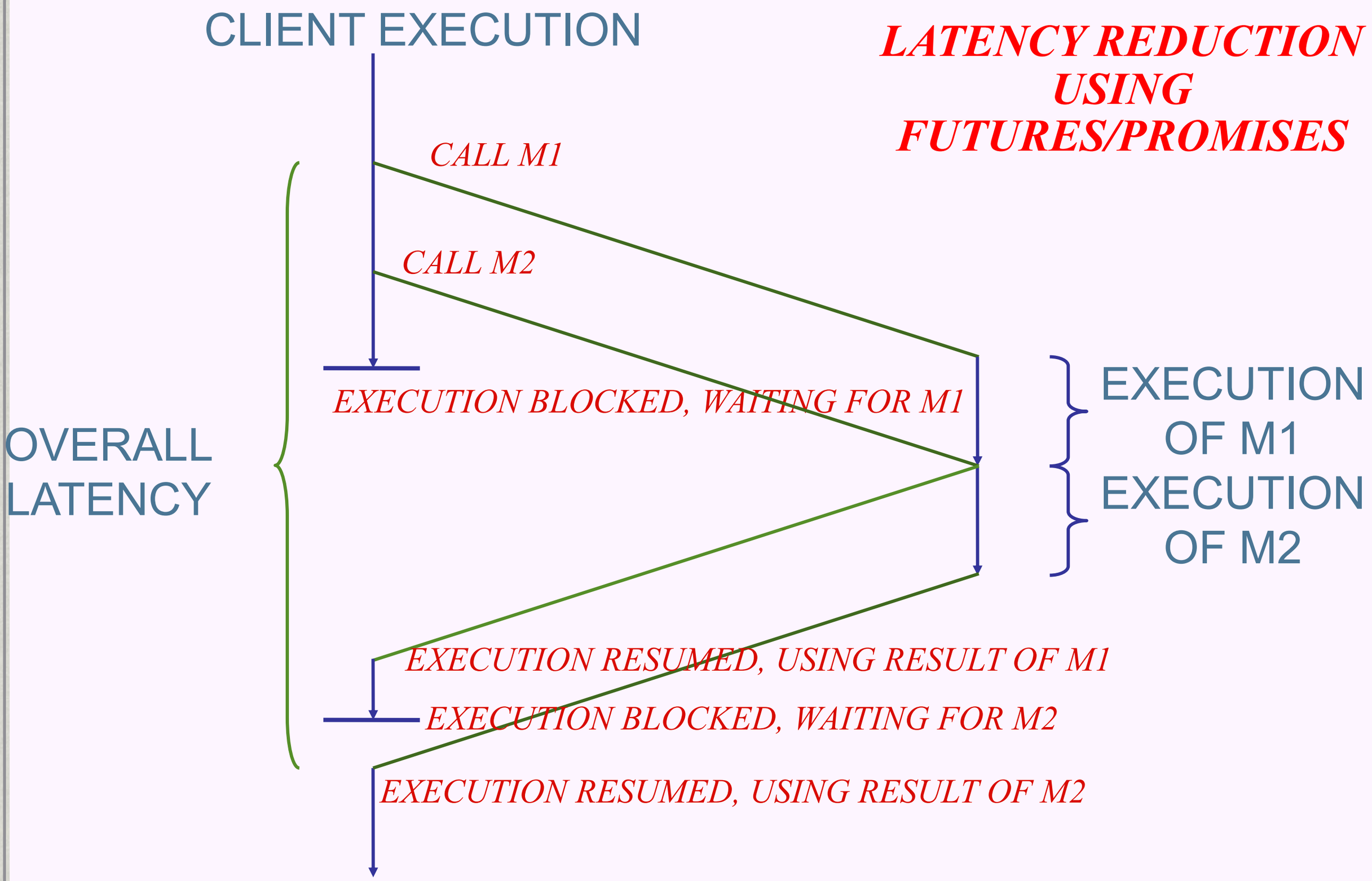
3

- 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, …)
```

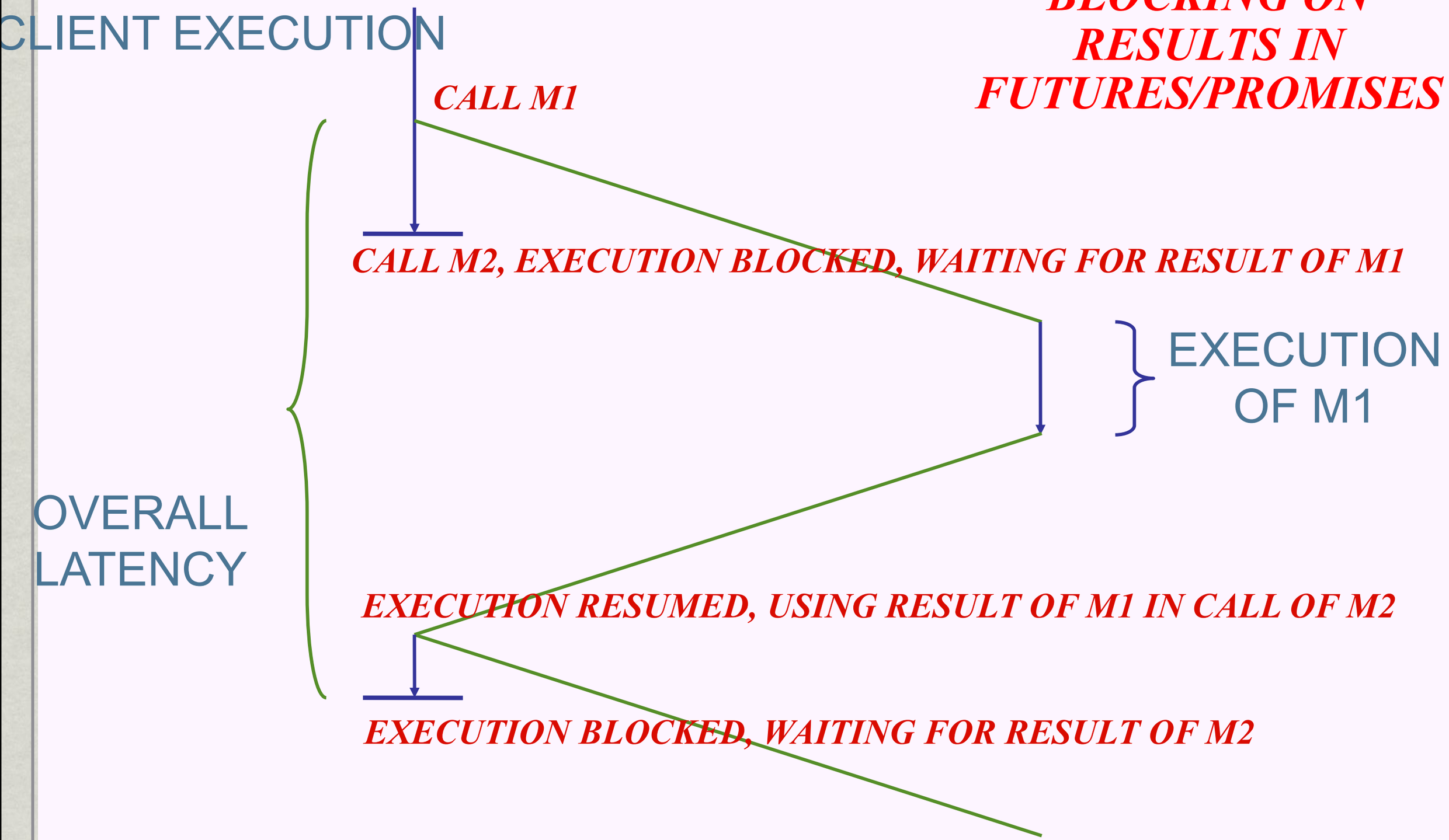
- With explicit claims, this must be re-written:

```
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

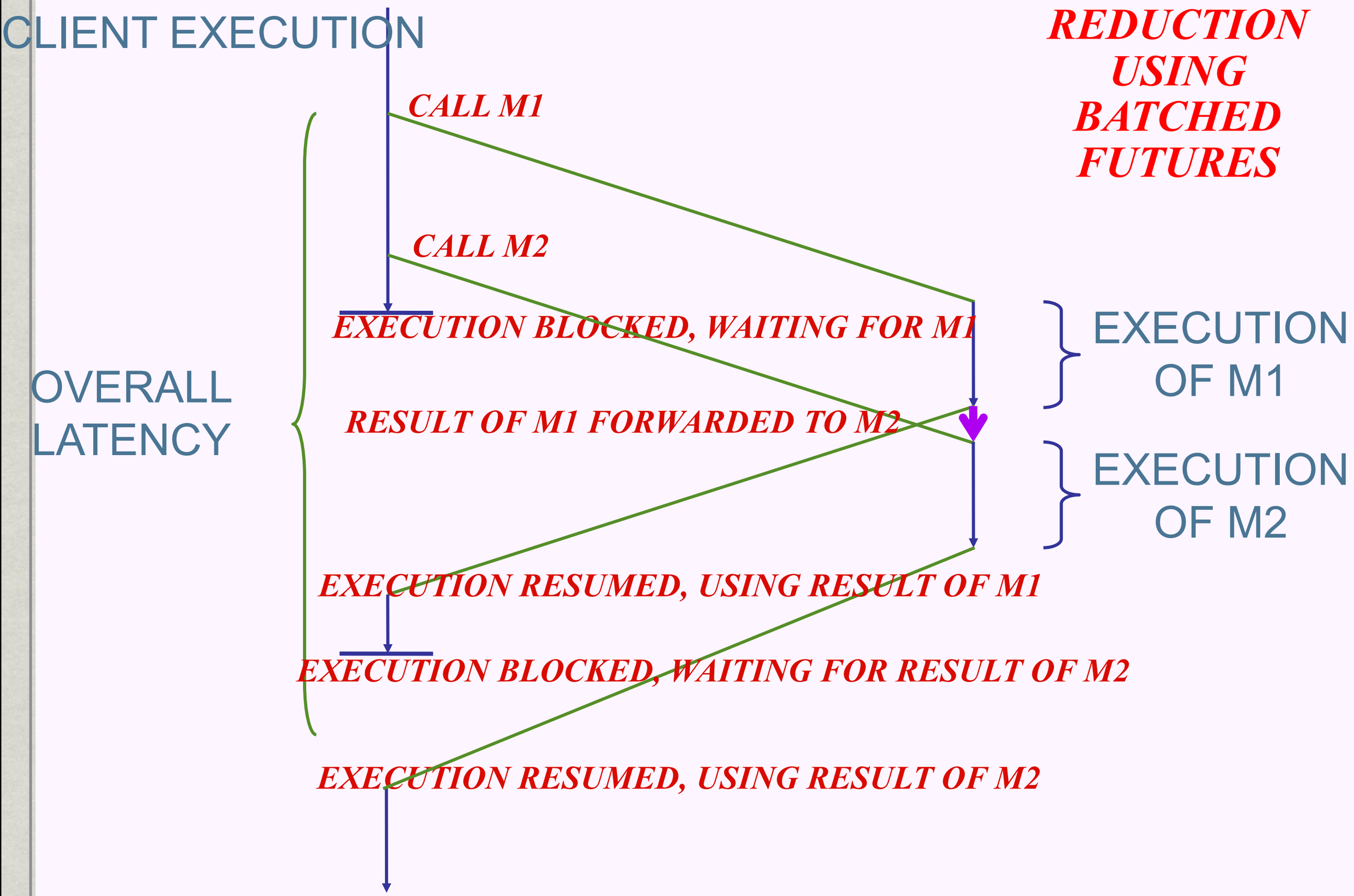
4

- 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.

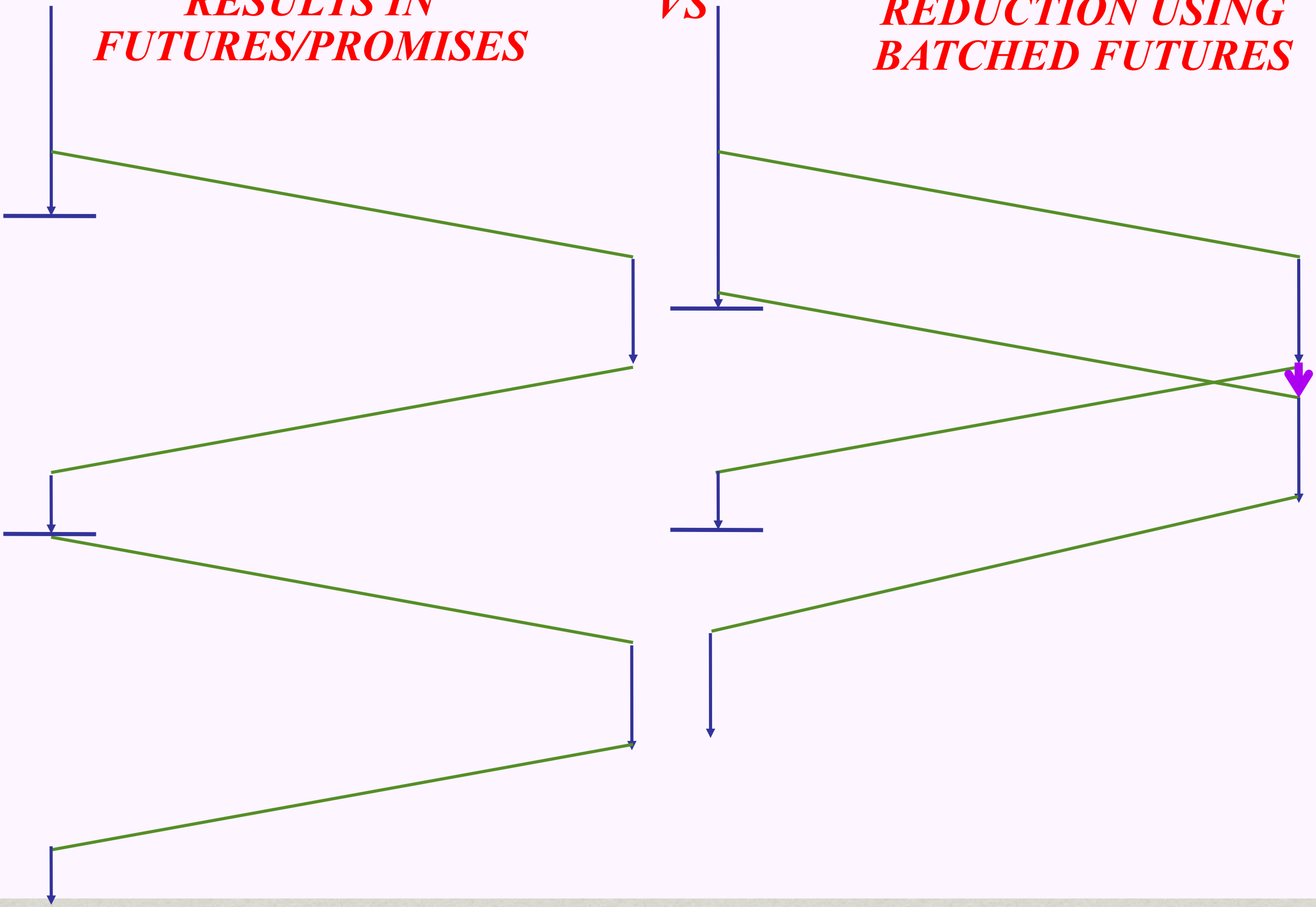
LATENCY REDUCTION USING BATCHED FUTURES



BLOCKING ON RESULTS IN FUTURES/PROMISES

VS

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

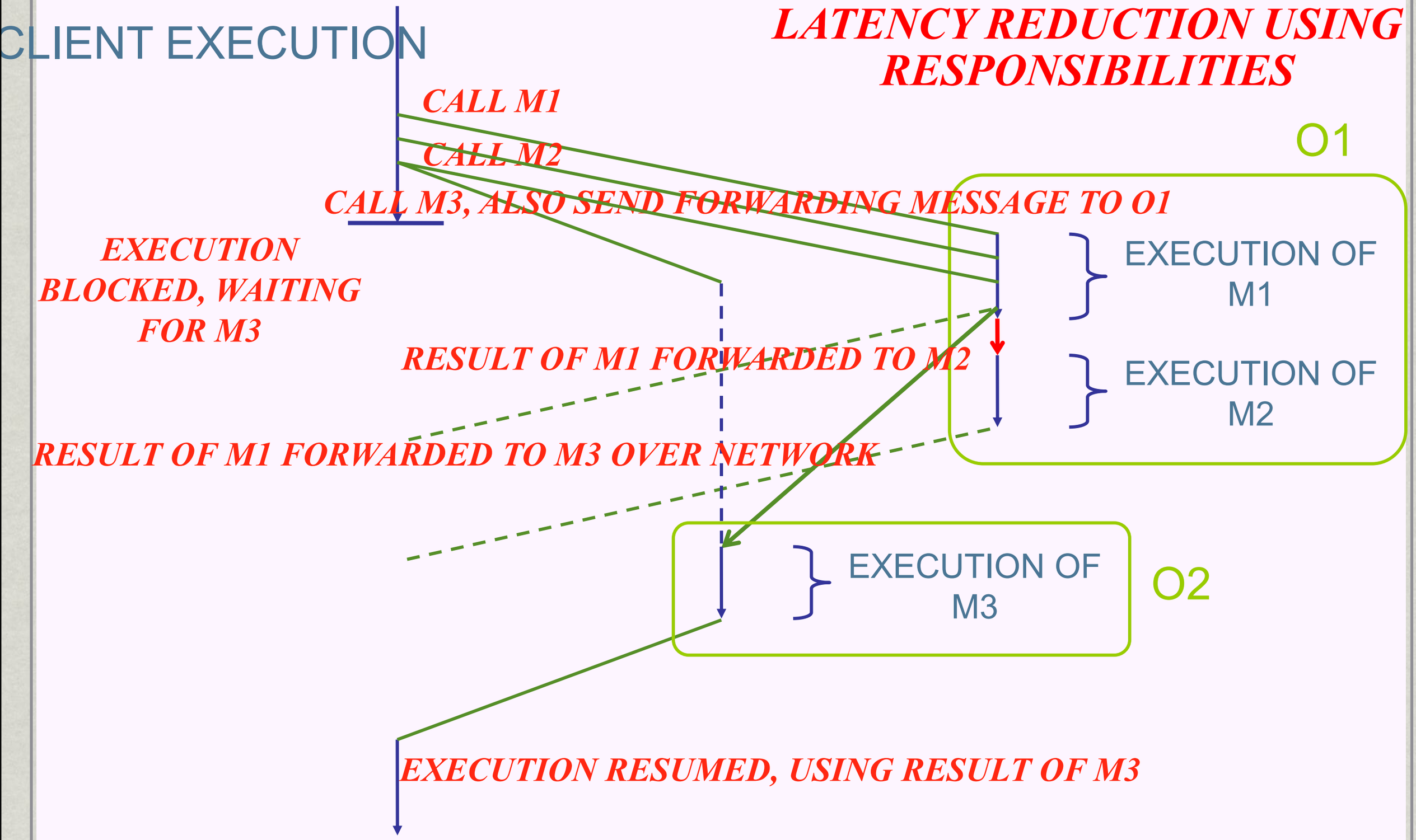
5

- 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:

- For example:
  res1 = O.m1(…)
  res2 = O.m2(…, res1 + 3 , …)
  or
  res2 = O.m2(…, O.m3(res1), …)

Key points

- There are many of latency hiding and reduction techniques
- We can analyse them to determine best, worst and average case performance
- When designing your system you will have new attributes to consider and determine it's worth in time and effort
  - Number of calls, dependencies, frequency, number of servers, blocking (critical or not)
- Most importantly, rationalising your choice of solution

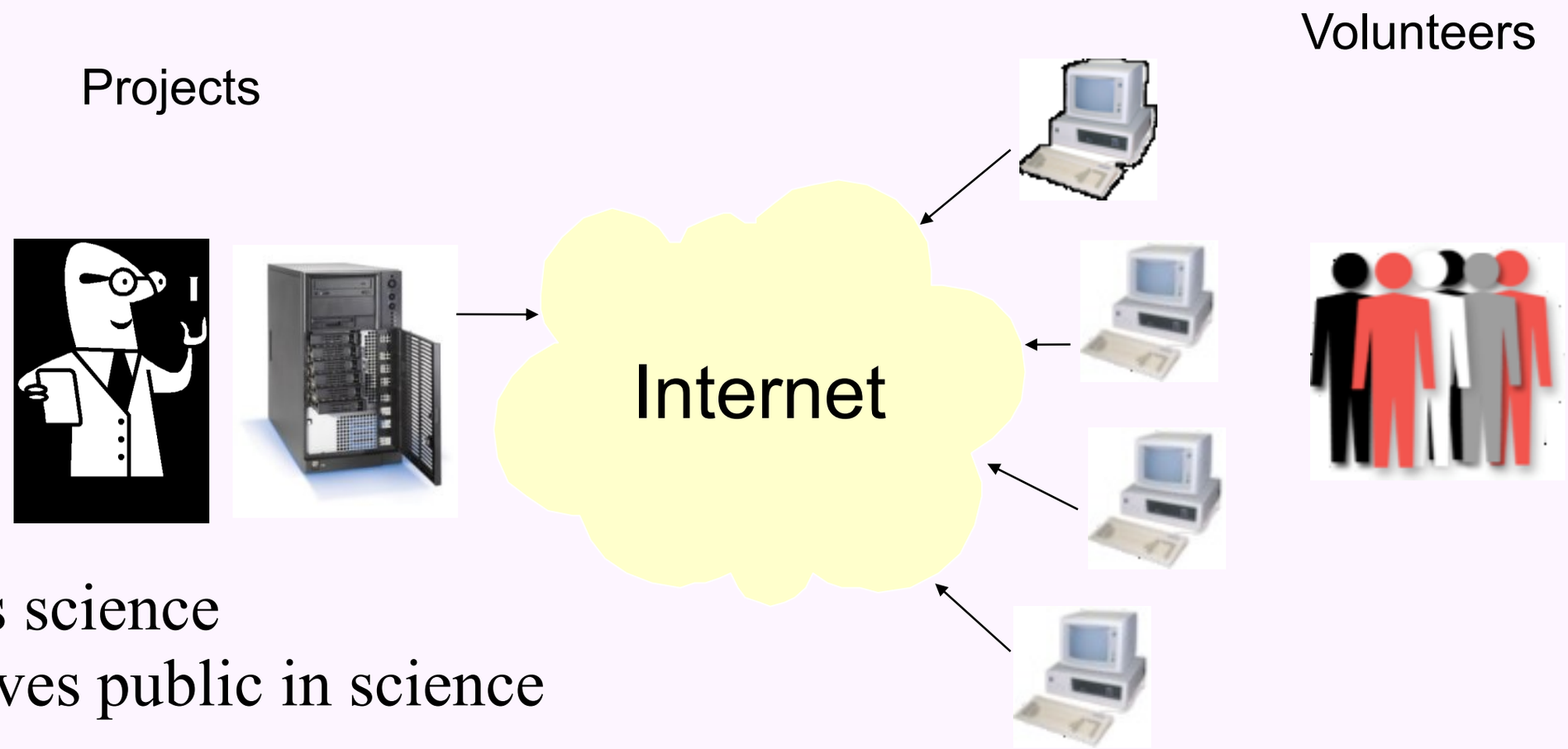
AN x64 PROCESSOR IS SCREAMING ALONG AT BILLIONS OF CYCLES PER SECOND TO RUN THE XNU KERNEL, WHICH IS FRANTICALLY WORKING THROUGH ALL THE POSIX-SPECIFIED ABSTRACTION TO CREATE THE DARWIN SYSTEM UNDERLYING OS X, WHICH IN TURN IS STRAINING ITSELF TO RUN FIREFOX AND ITS GECKO RENDERER, WHICH CREATES A FLASH OBJECT WHICH RENDERS DOZENS OF VIDEO FRAMES EVERY SECOND

BECAUSE I WANTED TO SEE A CAT JUMP INTO A BOX AND FALL OVER.

I AM A GOD.

Systems talk - volunteer computing

Volunteer Computing



- Helps science
- Involves public in science

SETI@home

- SETI (Search for Extraterrestrial Intelligence)
  - scientific area whose goal is to detect intelligent life outside Earth
  - radio SETI: uses radio telescopes to listen for narrow-bandwidth radio signals from space. Such signals are not known to occur naturally, so a detection would provide evidence of extraterrestrial technology.
- Data is divided into small chunks that are then deployed on host computers for processing; once done, results are sent back
- SETI client deployed on 3.5 million hosts (5 million since starting)
  - 226 countries, 50% outside US
  - 3.5 PFLOP actual performance

BOINC

- Berkeley Open Infrastructure for Network Computing
- Middleware for volunteer computing
- Open-source (LGPL)
- Application-driven
- Goals
  - lots of independent projects
  - support for diverse applications
  - client participation in multiple projects

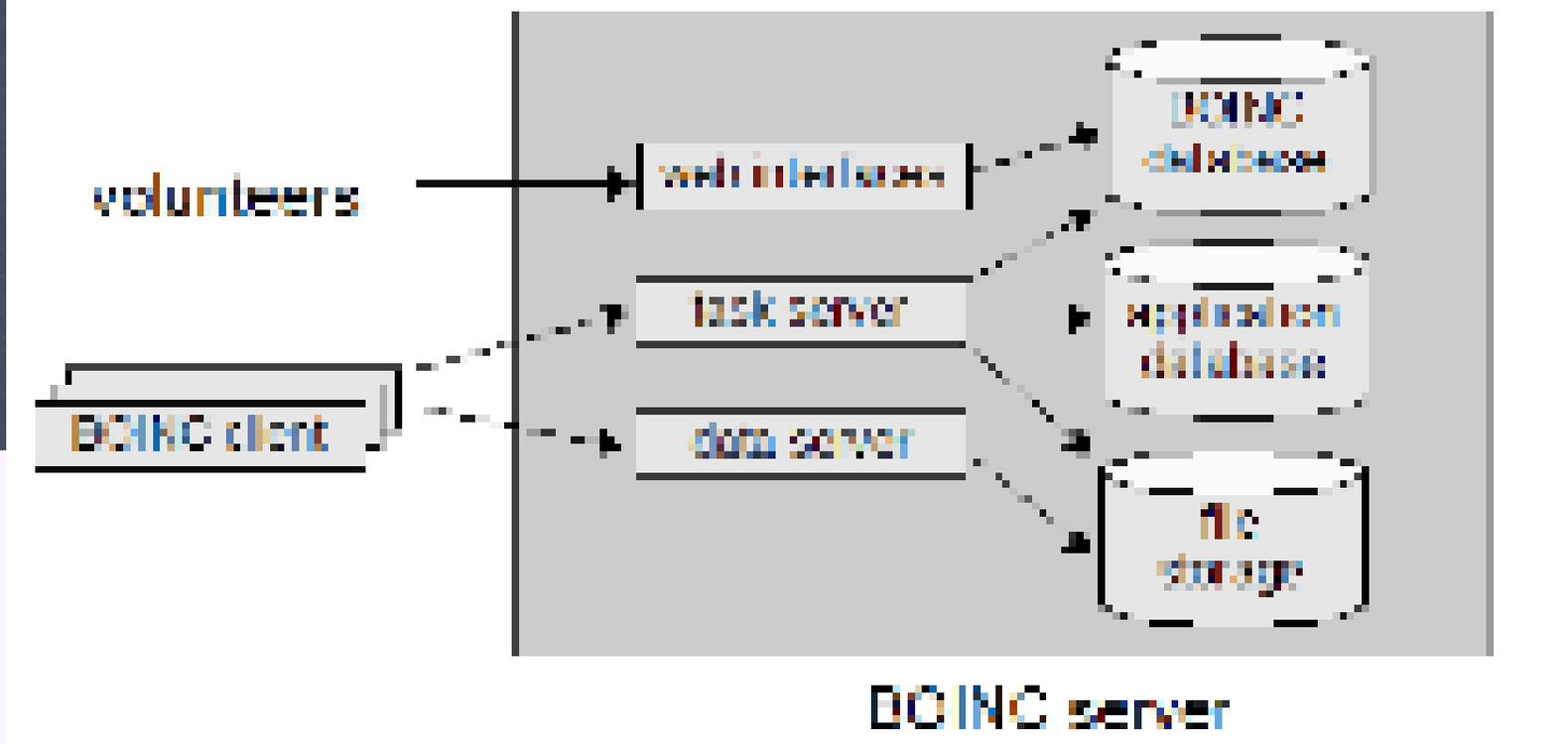
Projects

- Climateprediction.net - Oxford; climate change study
- Einstein@home - LIGO; gravitational wave astronomy
- Rosetta@home - U. Washington; protein study
- SETI@home - U.C. Berkeley; SETI
- LHC@home - CERN; accelerator simulation
- Africa@home - STI, U. of Geneva; malaria epidemiology
- IBM World Community Grid - several biomedical applications
- ...and about 30 others

How?

- Client-server application
- Servers: scheduling servers handle RPC calls, data handling servers store data
- Clients perform computations; results are uploaded to the server
- Backend server validates results
  - same task is executed on multiple client computers; results are compared

How?



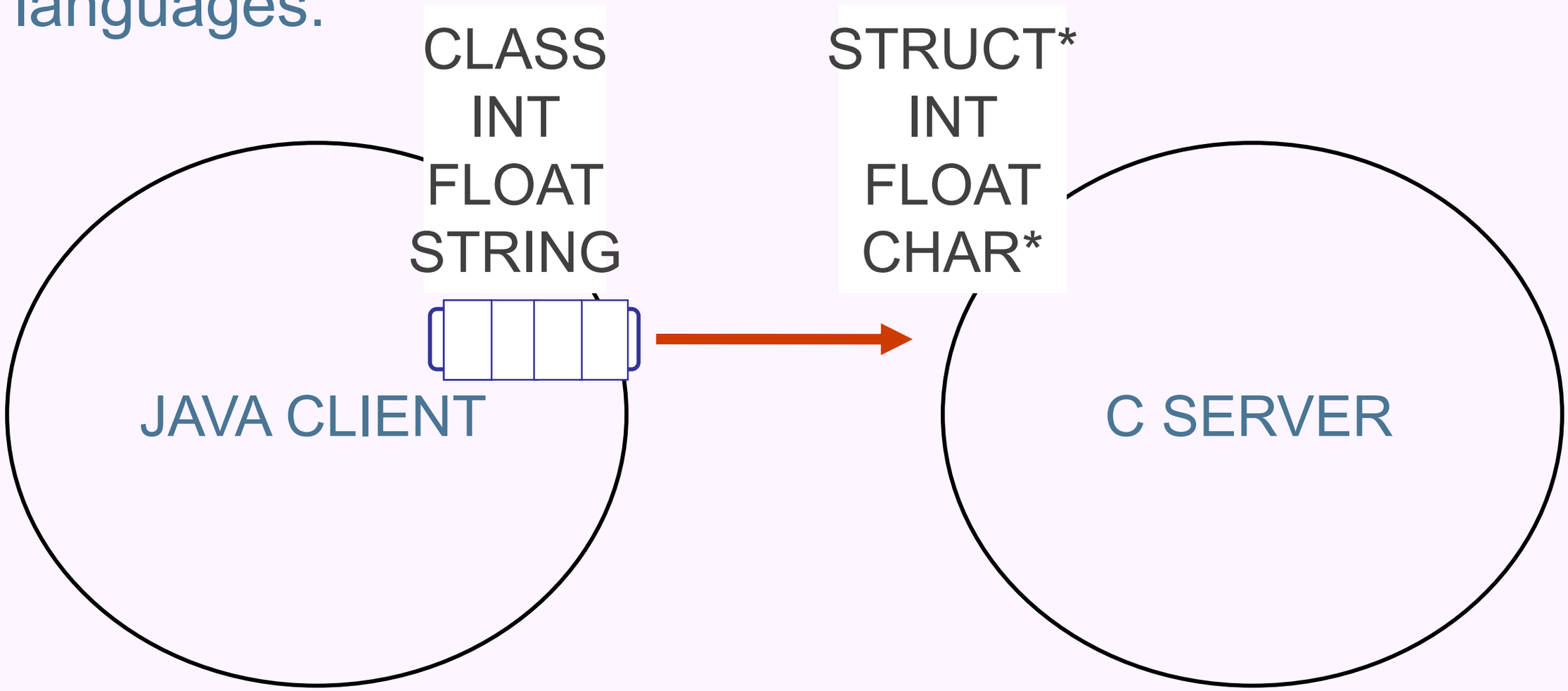
- Other features
  - homogeneous redundancy - same workunit is sent to computers with same platform
  - trickling - partial info is sent to the server before the unit completes
  - locality scheduling - workunits are sent to computers that have the necessary files
  - work distribution - workunit distributed according to computational power

Two more problems ...

- Communication
- Sending stuff across

The Problem

- Communication between different programming languages:



Linguistic Heterogeneity

- Distributed systems in which components are written in different programming languages.
- Why is this important?
- Distributed systems are rarely built from scratch:
  - Buy versus build – it may be possible to purchase commercial off-the-shelf distributed system components at lower cost than building equivalent components.
  - Legacy components – an organisation may have existing distributed system components, typically containing valuable data, which can't be replaced but must be part of some new distributed system.
- In both cases, the pre-existing components are most probably written in a diverse collection of programming languages.

Linguistic Heterogeneity

- Communication between different programming languages:
  - Each programming language has its own ***TYPE SYSTEM***.
  - The type system describes how a programming language represents program data.
- The problems:
  - Each type system may have a different representation for equivalent data.
  - Some type systems may have concepts that have no equivalent in some others.
- In order to send data between different languages, we need a bridge over these differences.

A Consensus Type System

- A ***CONSENSUS TYPE SYSTEM*** is:
  - A limited programming language.
  - Consisting only of type definition aspects.
  - Containing facilities to describe types common to a large range of programming languages.
- The most widely known consensus type system is ***CORBA IDL***:
  - CORBA = Common Object Request Broker Architecture.
  - IDL = Interface Definition Language.
- Note that the IDL – which we are interested in here – is only a part of CORBA, not the whole system.
  - And not only used by CORBA

IDL vs API

- IDLs are *not* the same as an API
- An API provides a service
- An IDL describes a service
- Both are intended to be used by multiple clients
  - This includes methods, classes, and all the above

CORBA IDL

- A type system for describing distributed object systems.
- Objects are described in terms of ***INTERFACES***, consisting of ***OPERATIONS***:
  - Very like interfaces in Java.
  - Operations are like method signatures – they have name and parameter types, but no code.
  - Unlike classes (and like Java interfaces), there is no data associated with CORBA interfaces.
  - Implementation details like method code and instance data are programming language specific and thus not represented in IDL.

CORBA IDL

- There is a CORBA ***BINDING*** for each common programming language:
  - How to export CORBA interfaces for programs written in the language.
  - How to represent CORBA interfaces within the language.

CORBA IDL Example

```
module BankExample {
  typedef float MoneyType;
  struct NameType {
    string first;
    string last;
  };
  interface BankAccount {
    MoneyType balance();
    MoneyType deposit(
      in MoneyType amount);
    MoneyType withdraw(
      in MoneyType amount);
  };
  interface CheckingAccount :
                          BankAccount {
    exception BadCheck {
      MoneyType fee;
    };
    MoneyType writeCheck(
      in MoneyType amount)
      raises (BadCheck);
  };
  interface BankManager {
    CheckingAccount openAccount(
      in NameType name,
      in MoneyType deposit,
      out MoneyType balance);
  };
};
```

Figure 1: Bank IDL Example

CORBA IDL Type System

- Aside from objects, described by user defined interfaces, IDL has a range of built-in types:
  - Numbers
  - Strings
  - Basic collections like arrays.
- The types of operation parameters may be either built-in types or interfaces (objects).

CORBA IDL Type System

- In the original version of CORBA, objects are passed by reference:
  - Unlike Java, there is no passing of objects by copy – sending complex data structures around is difficult.
  - There is an “objects by value” extension.
  - Mobile code is not really possible in a heterogeneous system – objects are essentially stationary, in a fixed place on the network.
- Parameters of built in types are passed by copying.

XML

- ***XML***, eXtensible Markup Language, is a means to described structured data in textual form.
- There is a a type system for XML – ***XML-SCHEMA***:
  - This can describe most data types in common programming languages.
- Communication of XML data is normally pass-by-copy:
  - Each programming language has a a programming interface for manipulating XML data.
- XML entities can be referenced with URL-like links (Xlink/Xpointer):
  - This supports a form of call-by-reference semantics.

XML

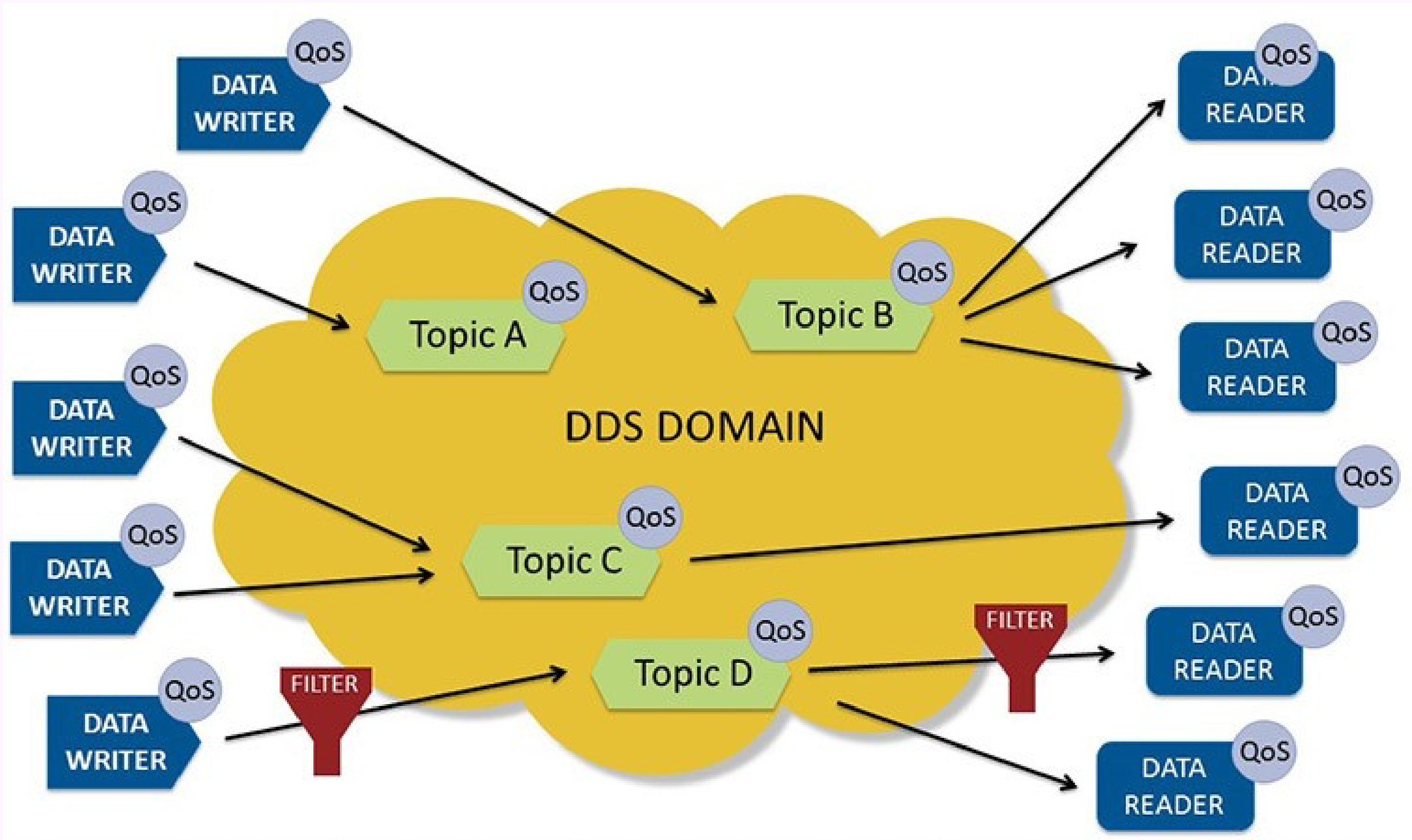
- Finally, there are XML facilities for describing service (somewhat like an object) interfaces (WSDL – Web Service Description Language):
  - The XML SOAP (originally Simple Object Access Protocol) is an XML means for describing how to call operations in services.

DDS

- Data Distribution Service
- Data-centric, many-to-many, message-passing communication, not RPC
- Like CORBA, is an OMG (open) standard for building distributed systems
  - Also uses IDL, but not Interfaces
  - RTPS
- Provides significant support and services, heterogeneous computing
- Popularised by USN: PEO IWS, AEGIS

DDS: Topics

Topic = {Name, Type, QoS



DDS: Services

- Discovery: Topics, Readers, Writers. Matching.
- Persistence: keep values after end of Writer lifetime
- Filtering: receive only a subset of samples
- Quality of Service

- Completely de-centralised.

DDS: Quality of Service

- A means of controlling performance/cost/semantic trade-off provided to application
  - Durability, Liveliness, Persistence
  - Reliability, Availability, Presentation Order
  - Deadline
  - Ownership
  - History, Resource Limits, etc
- A huge complexity-trap for the unwary/unprepared

Shortcomings

- IDL is a lot less expressive / extensible than XSD
- CORBA and DDS use IDL as their presentation grammar
  - They combine a communication system with a presentation system
  - Might be nicer if they were separate
- Difficult to build extensible, evolvable systems in CORBA/DDS or Java RMI

Recap

- Different computational and language models for building distributed systems
- Different ways of implementing openness & heterogeneity: interface specs, grammars
- Different tradeoffs wrt transparency, reliability, expressiveness, latency, efficiency, extensibility, scalability, etc.

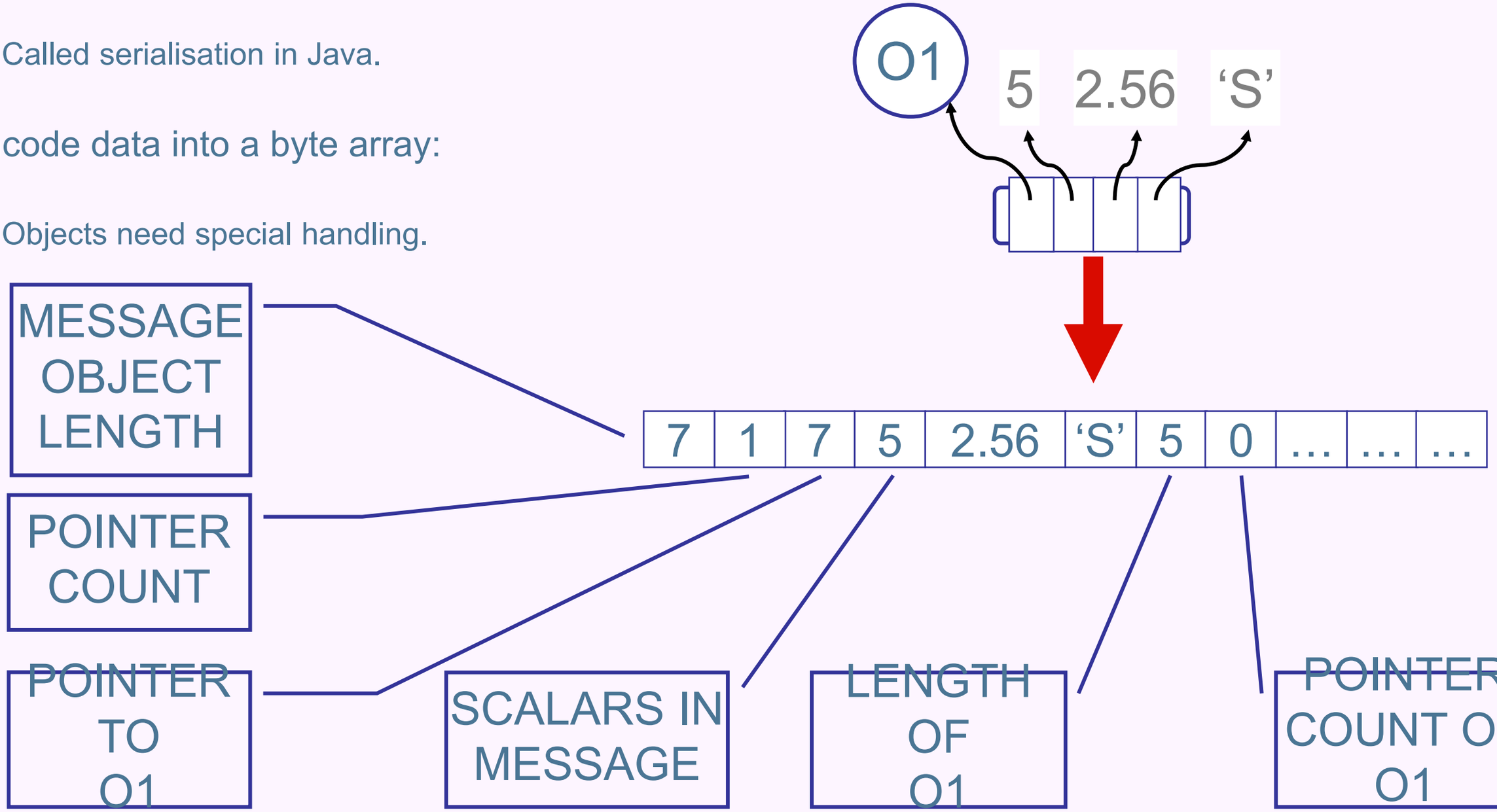
	Sockets	MPI	Java RMI	XMLRPC	REST	CORBA	DDS
Protocol	TCP/UDP	MPI	JRMP	SOAP/XML	HTTP/JSON	IIOP	RTPS
Grammar	None	None	Java interface	XSD/WSDL	N/A	IDL	IDL
Comm Primitive	Stream or Datagram	Message	RPC	RPC	RPC	RPC	Sample
Pattern	Point-Point Multicast/Broadcast	Point-Point, Broadcast	Point-Point	Point-Point	Client-Server	Point-Point	Many-Many
Pointers	No	No	Yes	URLs	URLs	Yes	No
Reliable	Optional	Yes	Yes	Yes	Yes	Yes	Optional
FIFO	Optional	Optional	Yes	Yes	Yes	Yes	Optional
Fault Tol	Manual	No	No	No	No	No	Optional

The problem

- Sending stuff across!

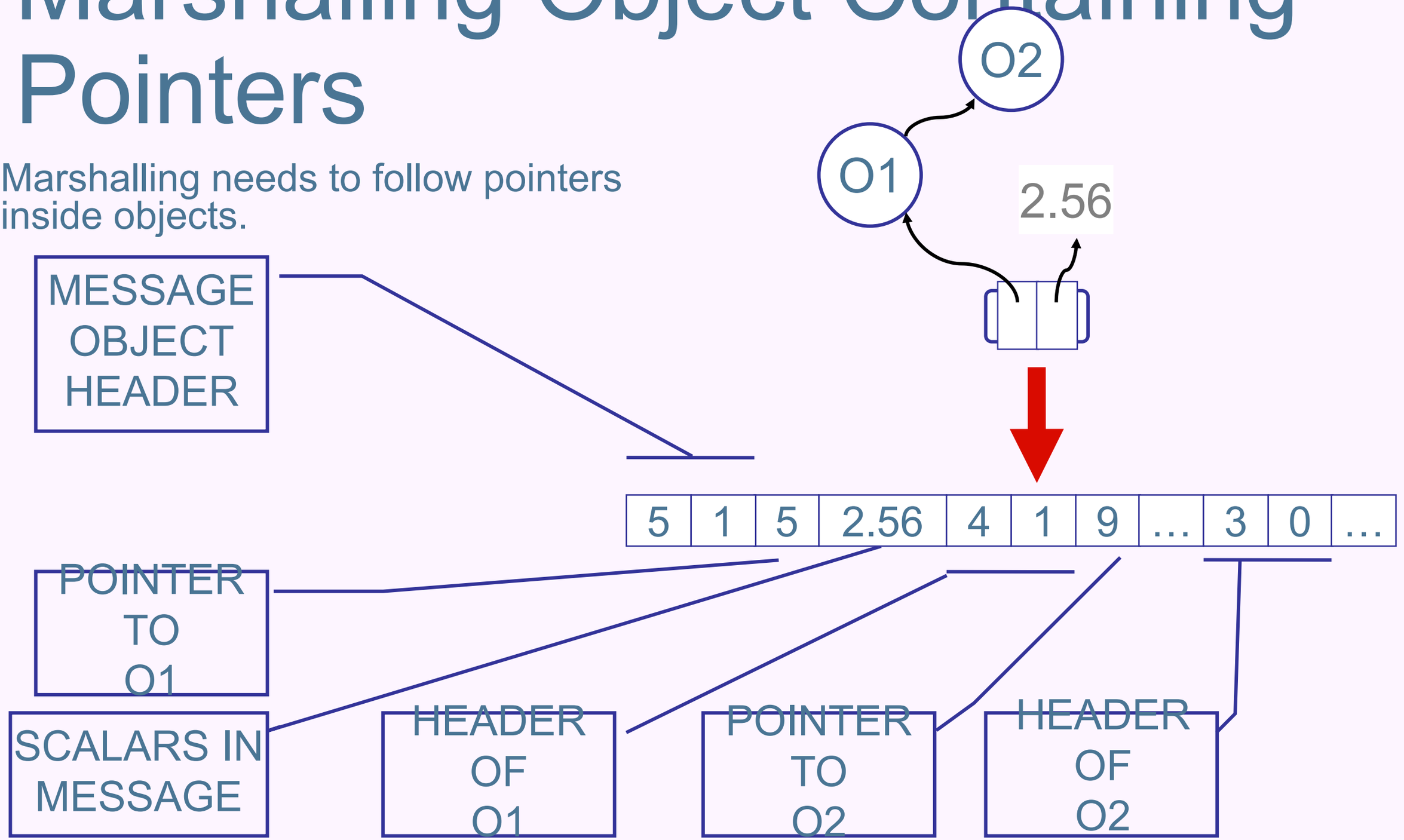
Marshalling

- 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.

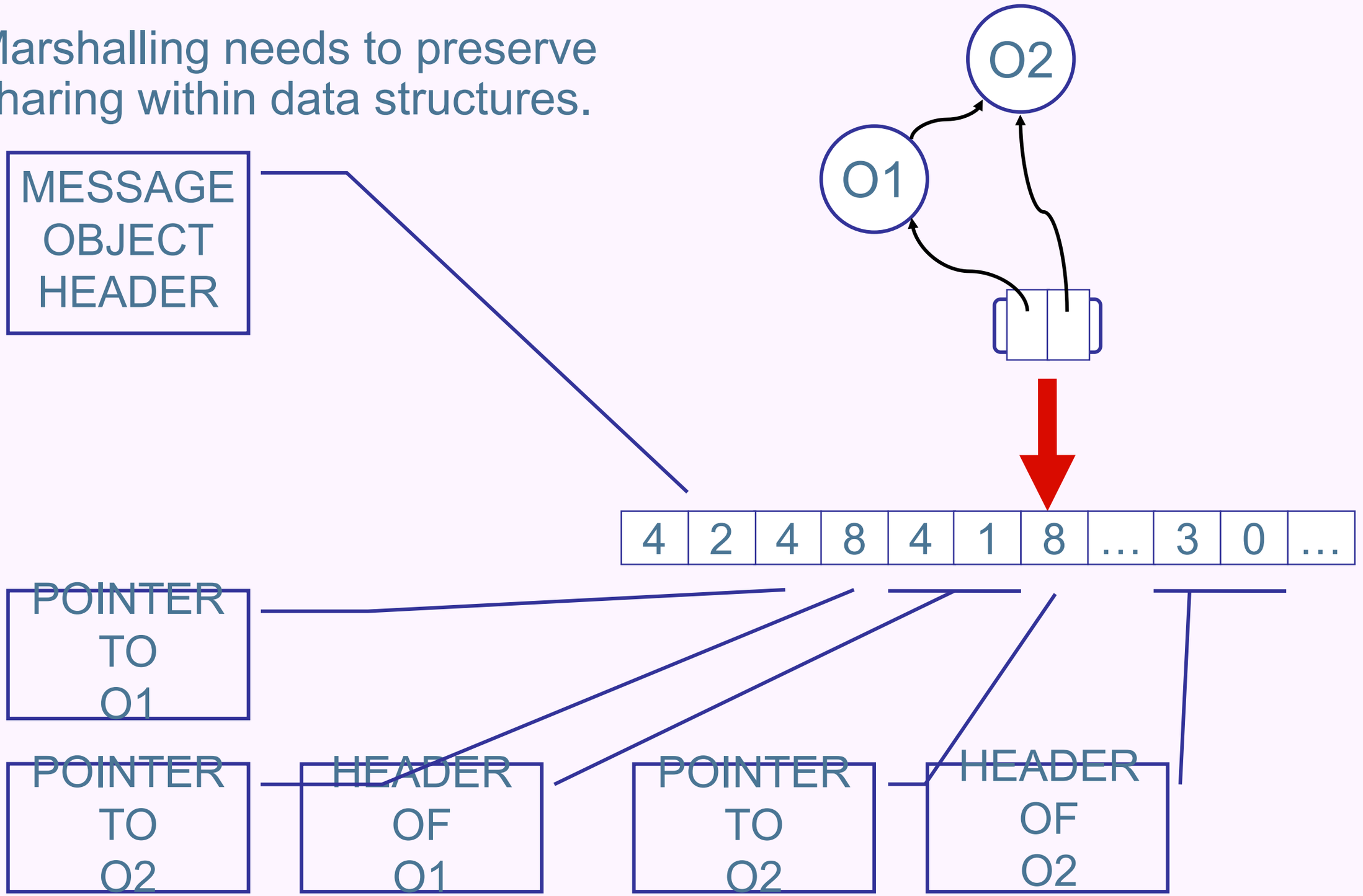


Marshalling Object Containing Pointers

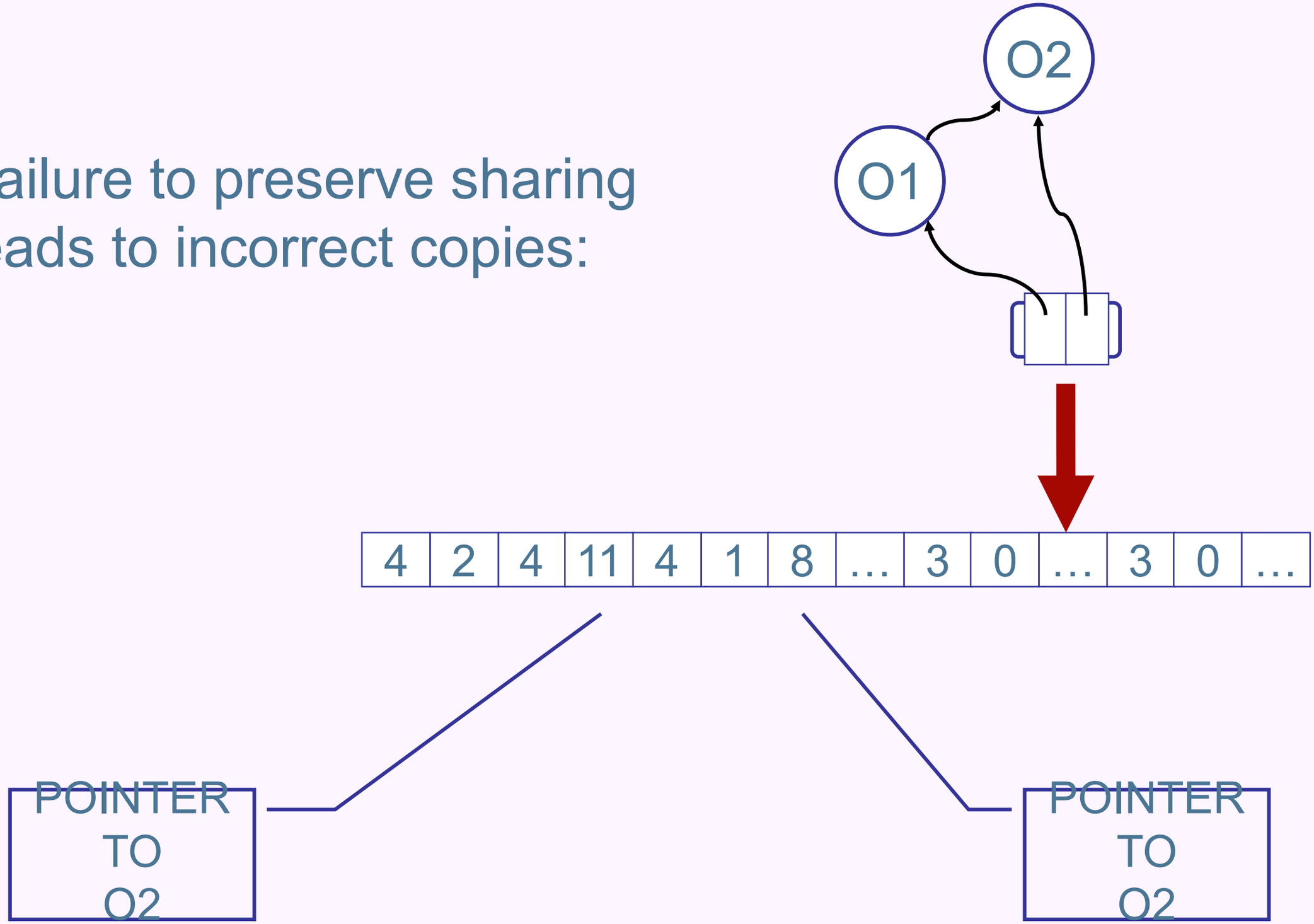
- Marshalling needs to follow pointers inside objects.



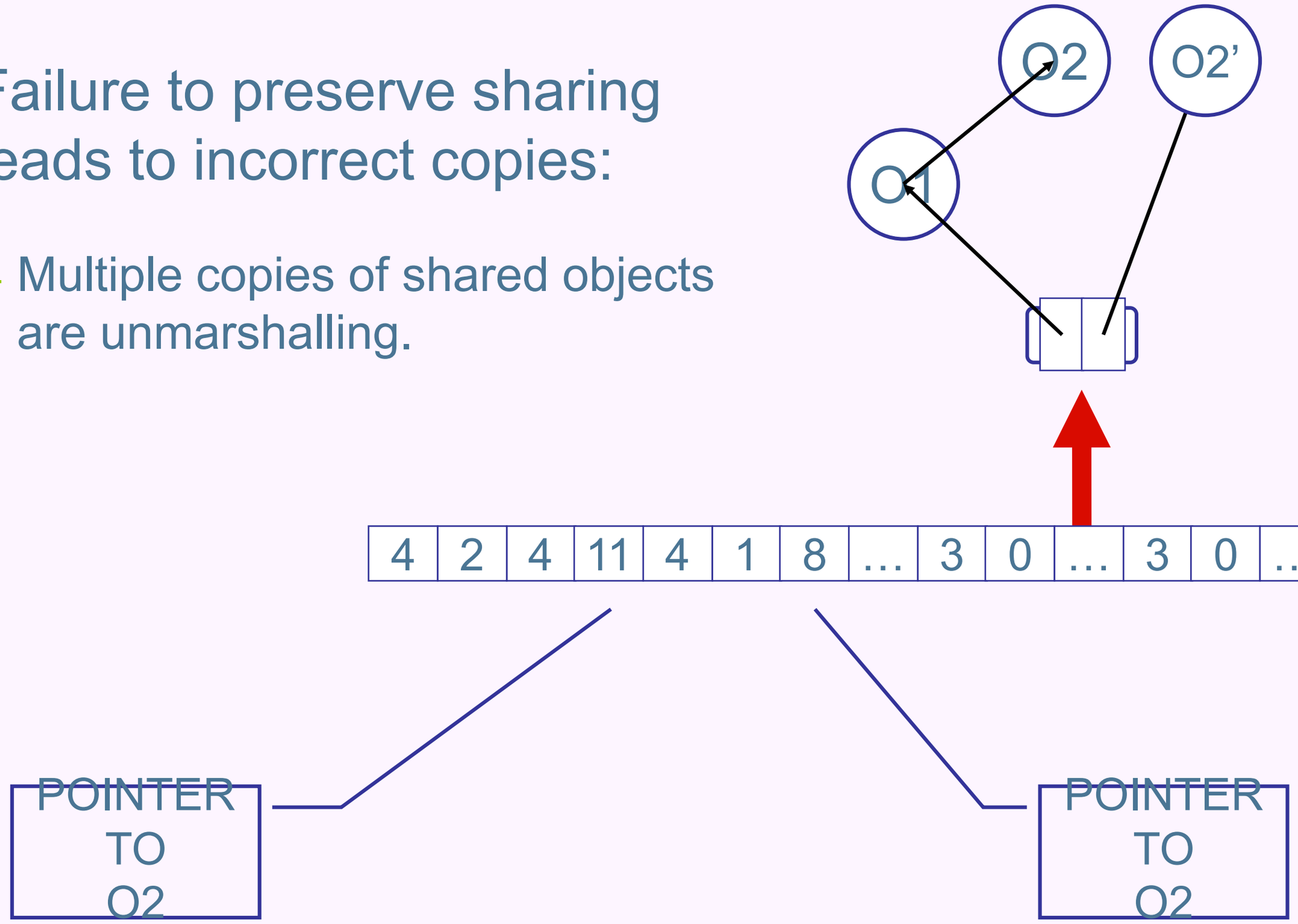
- Marshalling needs to preserve sharing within data structures.



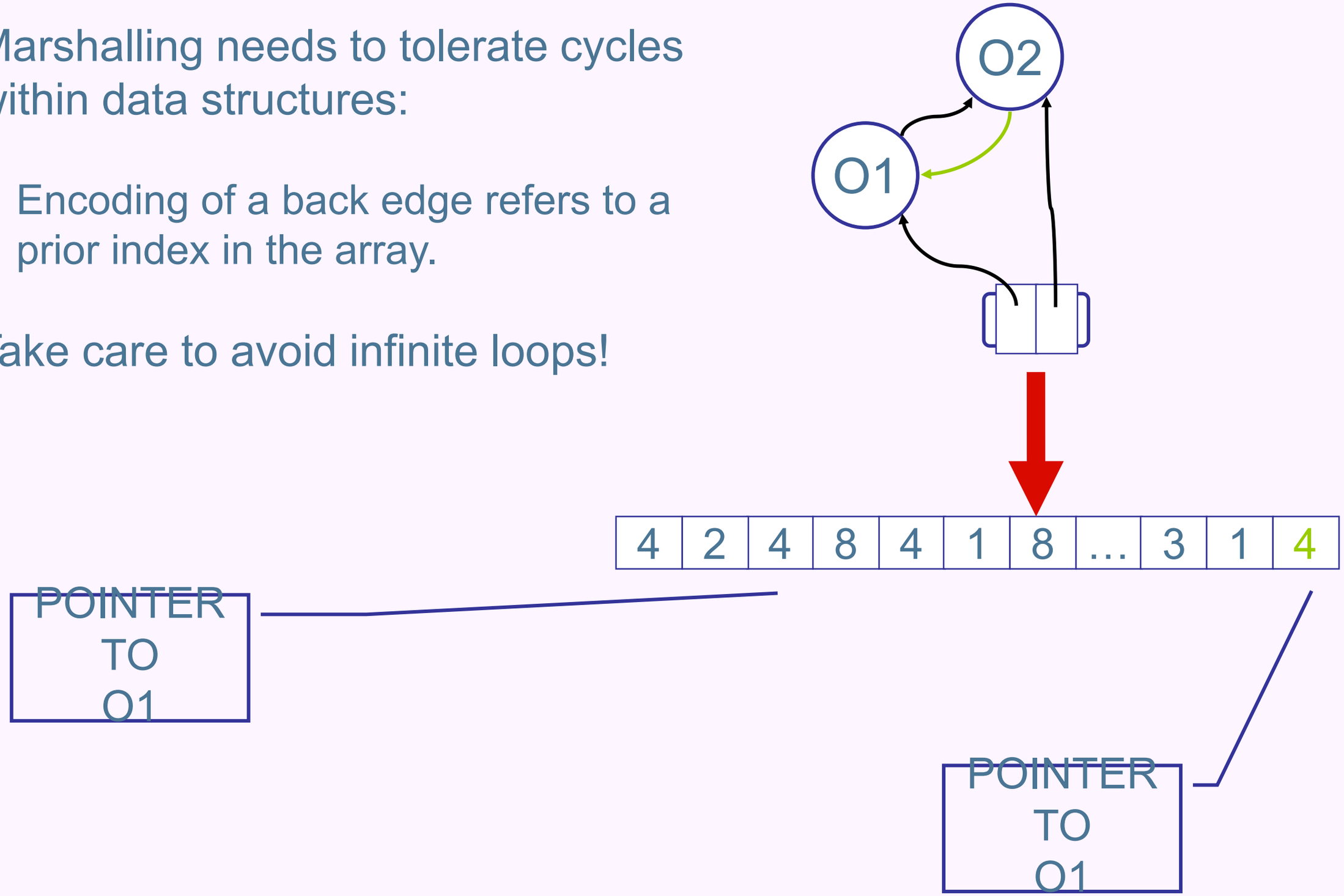
- Failure to preserve sharing leads to incorrect copies:



- 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!



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 DCG 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.