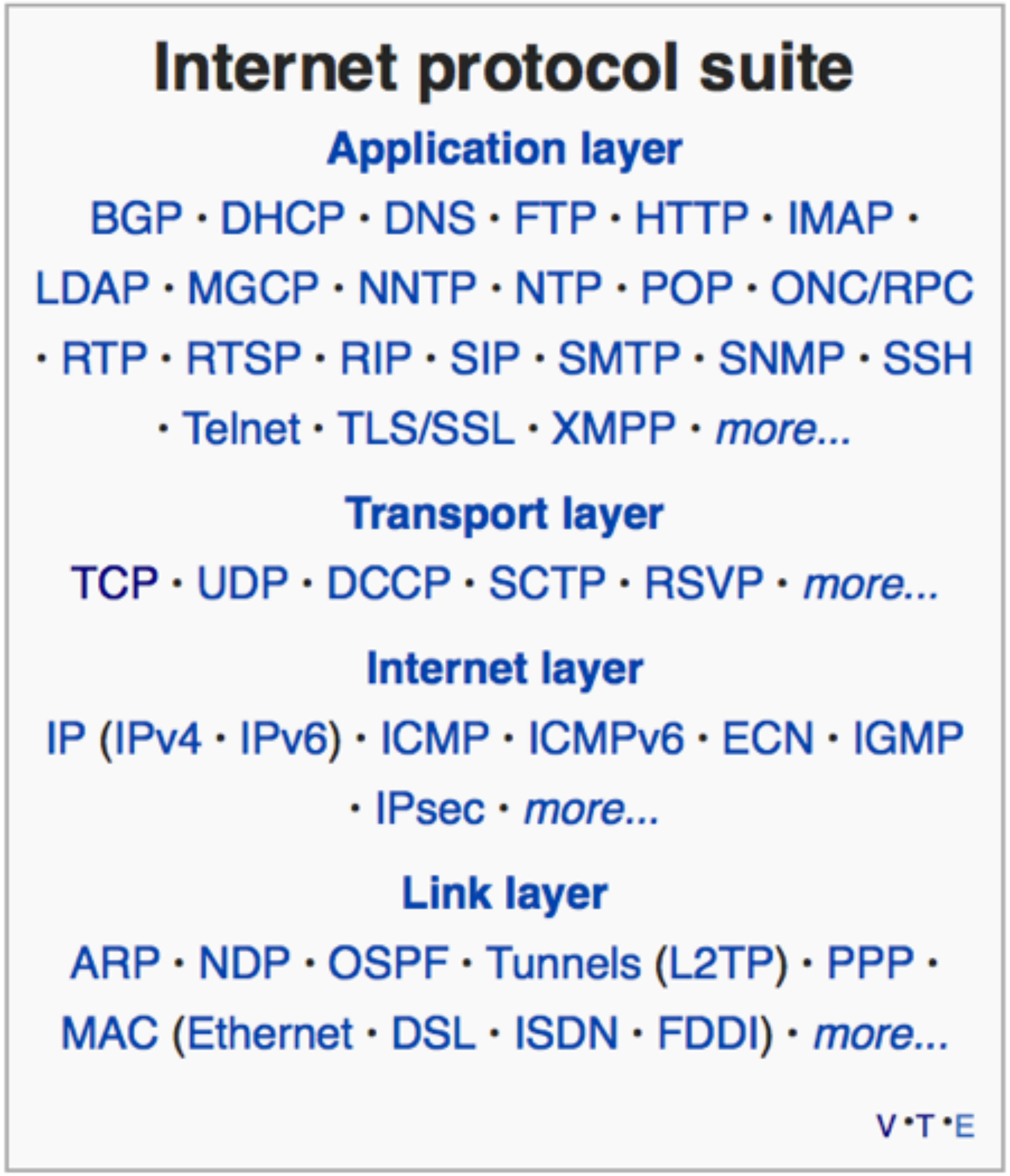
Protocols
The Glue for Applications

Torben Hoffmann
CTO @ Erlang Solutions
torben.hoffmann@erlang-solutions.com
@LeHoff

Why are we here?



Internet protocol suite

Application layer

BGP · DHCP · DNS · FTP · HTTP · IMAP · LDAP · MGCP · NNTP · NTP · POP · ONC/RPC · RTP · RTSP · RIP · SIP · SMTP · SNMP · SSH · Telnet · TLS/SSL · XMPP · *more...*

Transport layer

TCP · UDP · DCCP · SCTP · RSVP · *more...*

Internet layer

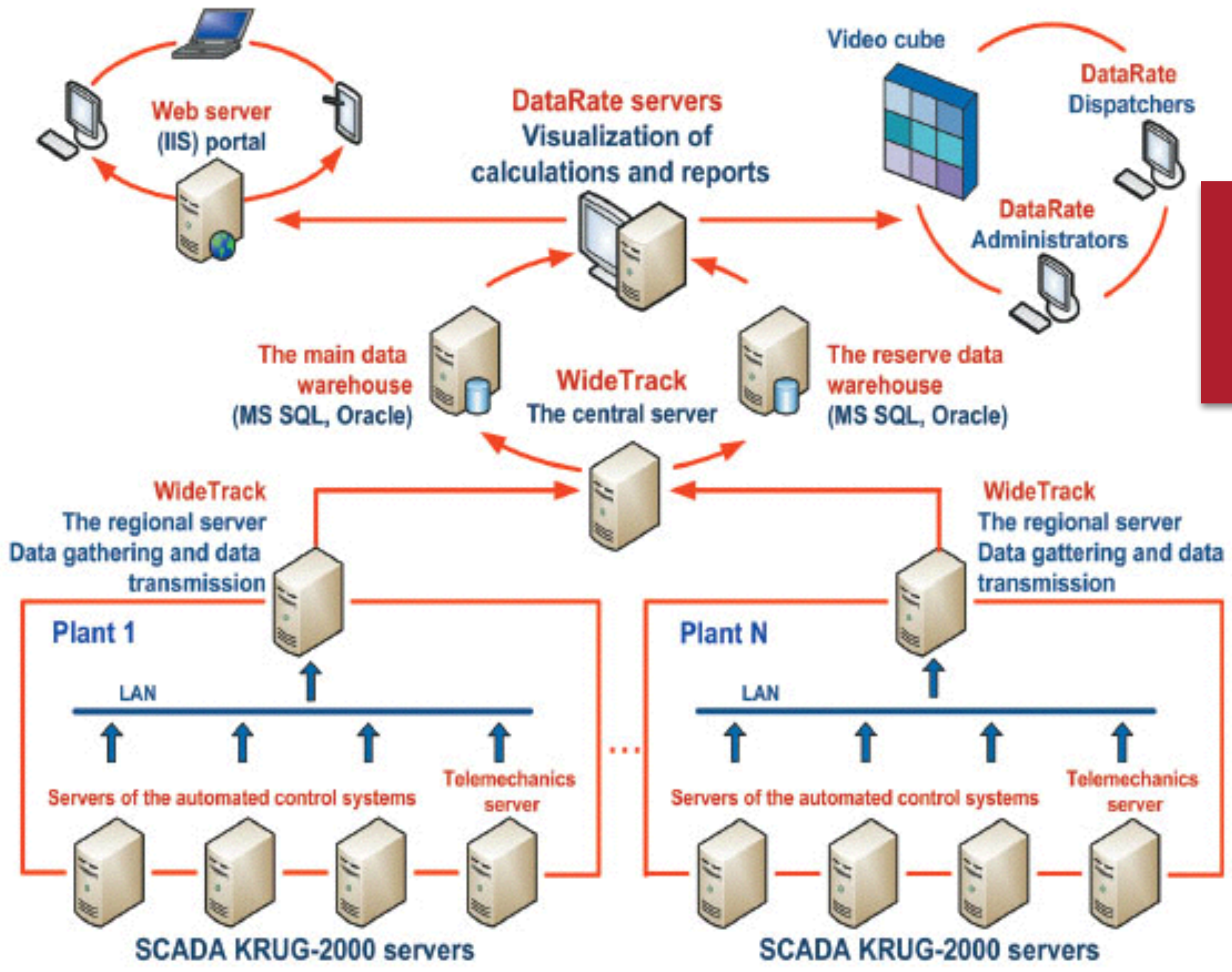
IP (IPv4 · IPv6) · ICMP · ICMPv6 · ECN · IGMP · IPsec · *more...*

Link layer

ARP · NDP · OSPF · Tunnels (L2TP) · PPP · MAC (Ethernet · DSL · ISDN · FDDI) · *more...*

V · T · E

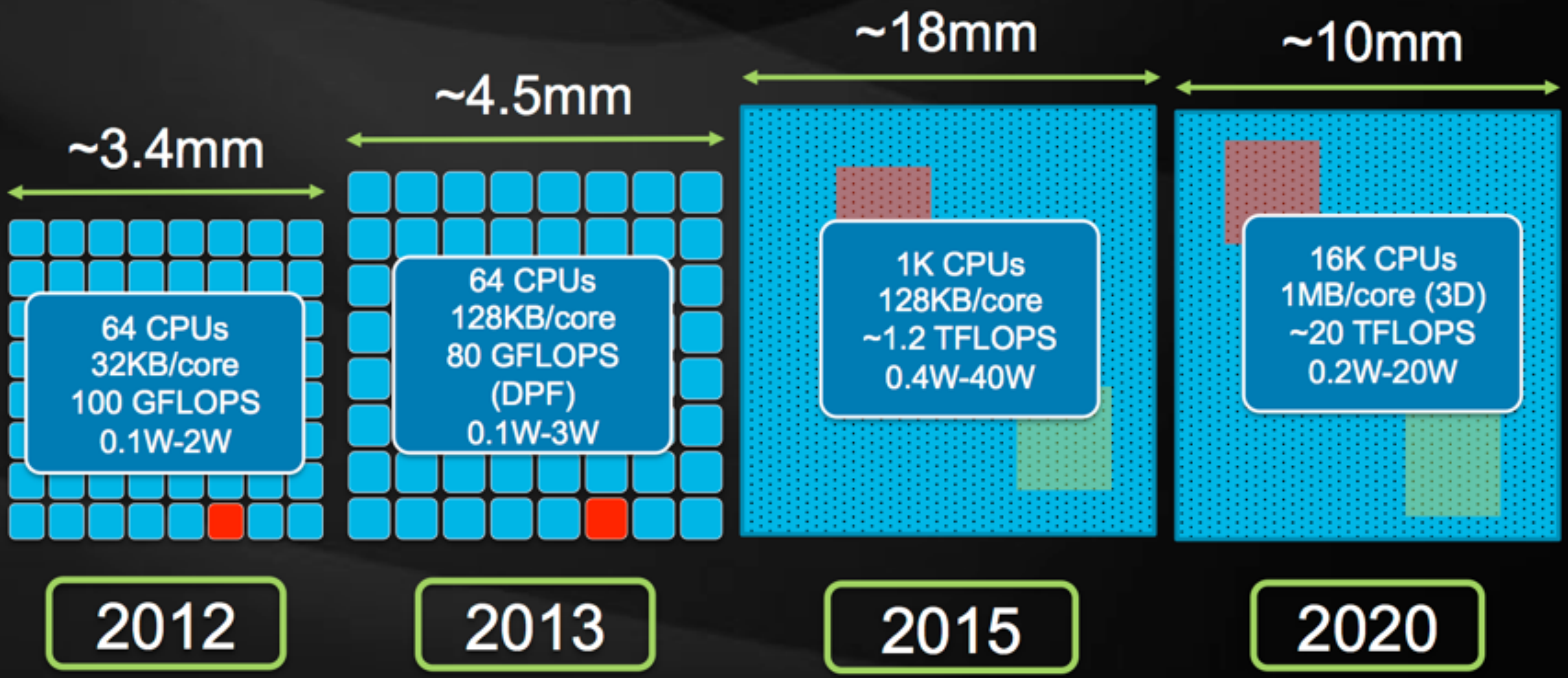
Distributed Systems



How likely is it that this will "just work"?

source: http://www.krug-soft.com/297.html

Epiphany Roadmap



adapteva

How often does WhatsApp have a failure?

WhatsApp MTBF

>600 machines

Assume failure rate of 1 in 2 years

$$MTBF = \frac{1}{1/2 + ... + 1/2} = 1/300a \approx 29h$$

1 machine going down daily!!

* http://www.abeacha.com/NIST_press_release_bugs_cost.htm

Failure is unavoidable

Global cost of IT failures

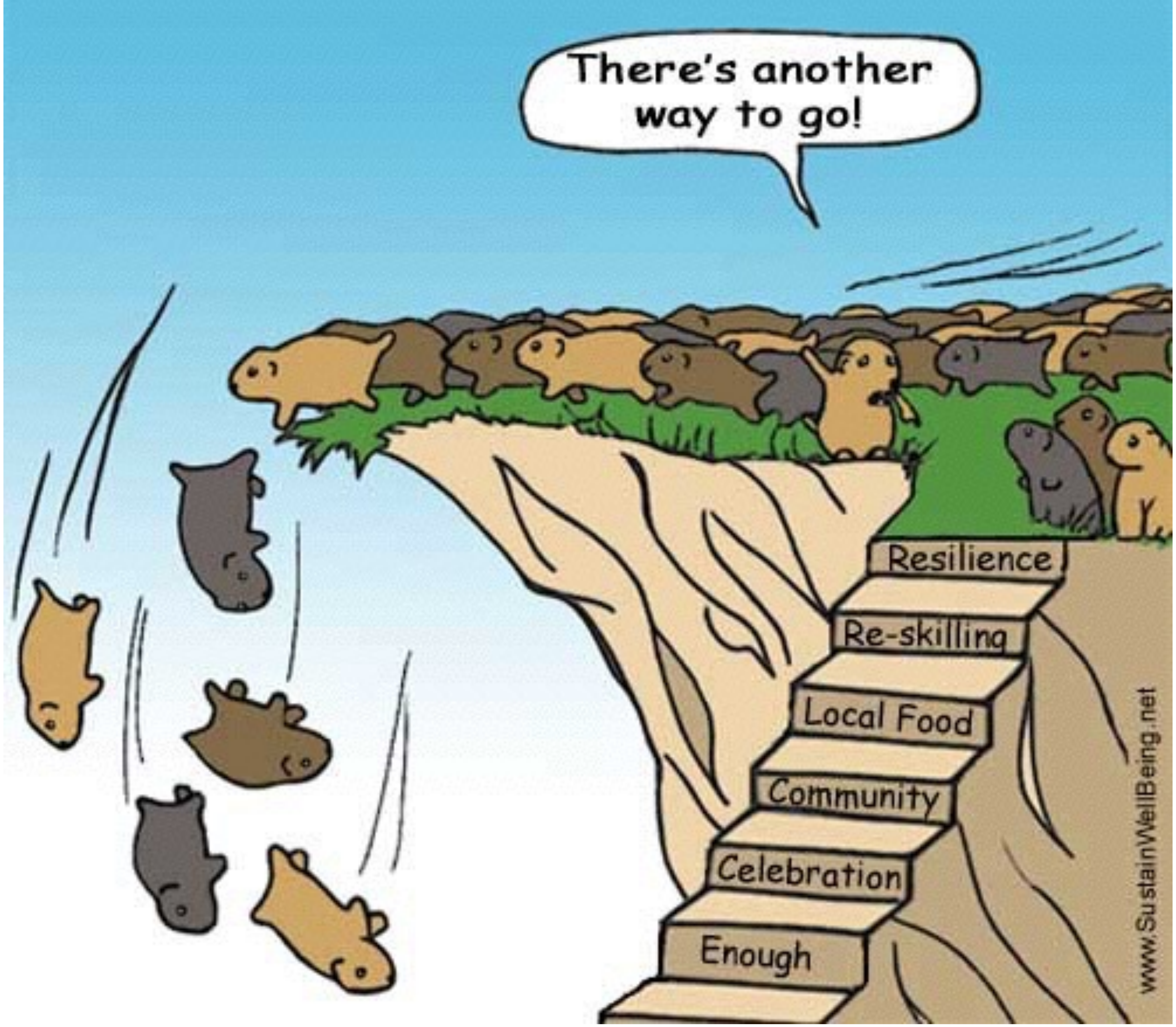
$3 Trillion

Annually

(Gene Kim and Mike Orzen)

source: http://www.zdnet.com/article/worldwide-cost-of-it-failure-revisited-3-trillion/

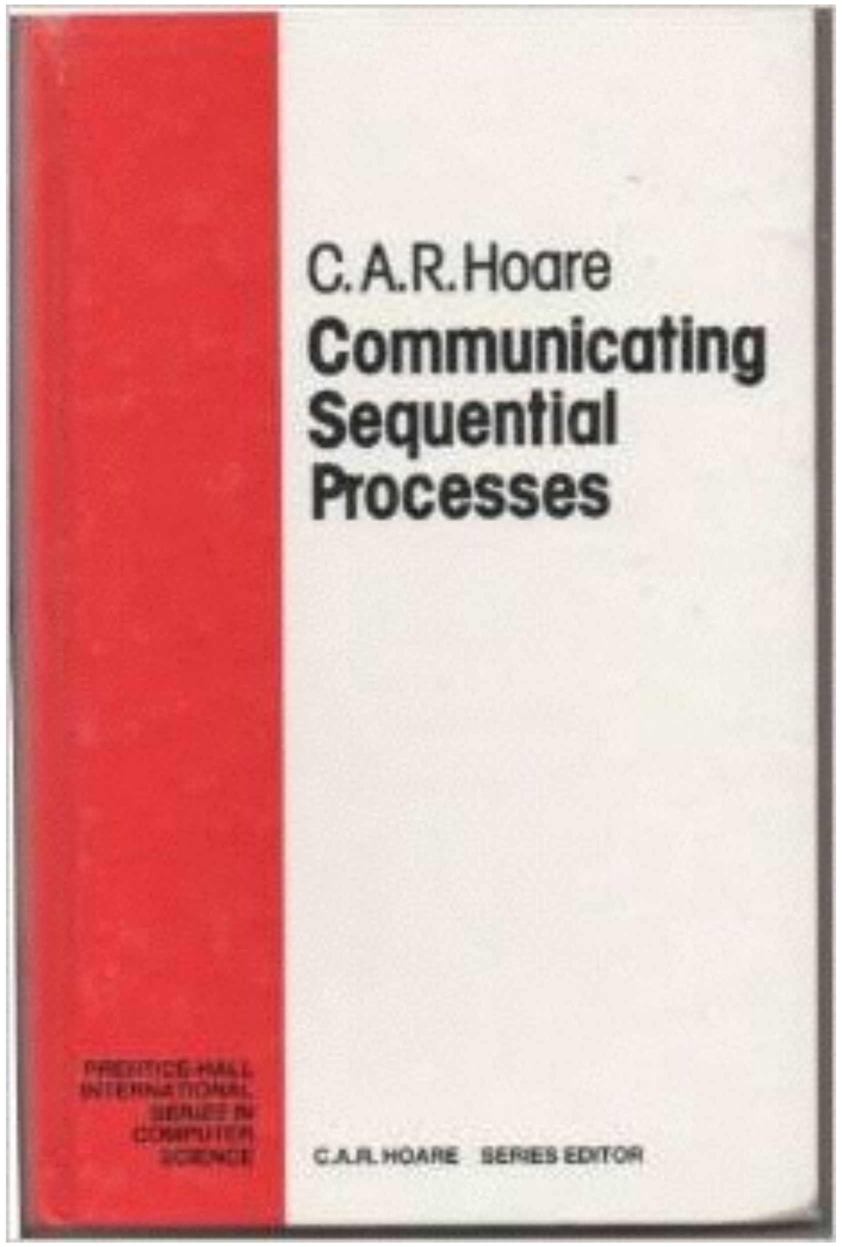
The thinking it took to get us into this mess is not the same thinking that is going to get us out of it.



Source: http://www.sustainwellbeing.net/lemmings.html

Methodology & Technology

Protocols

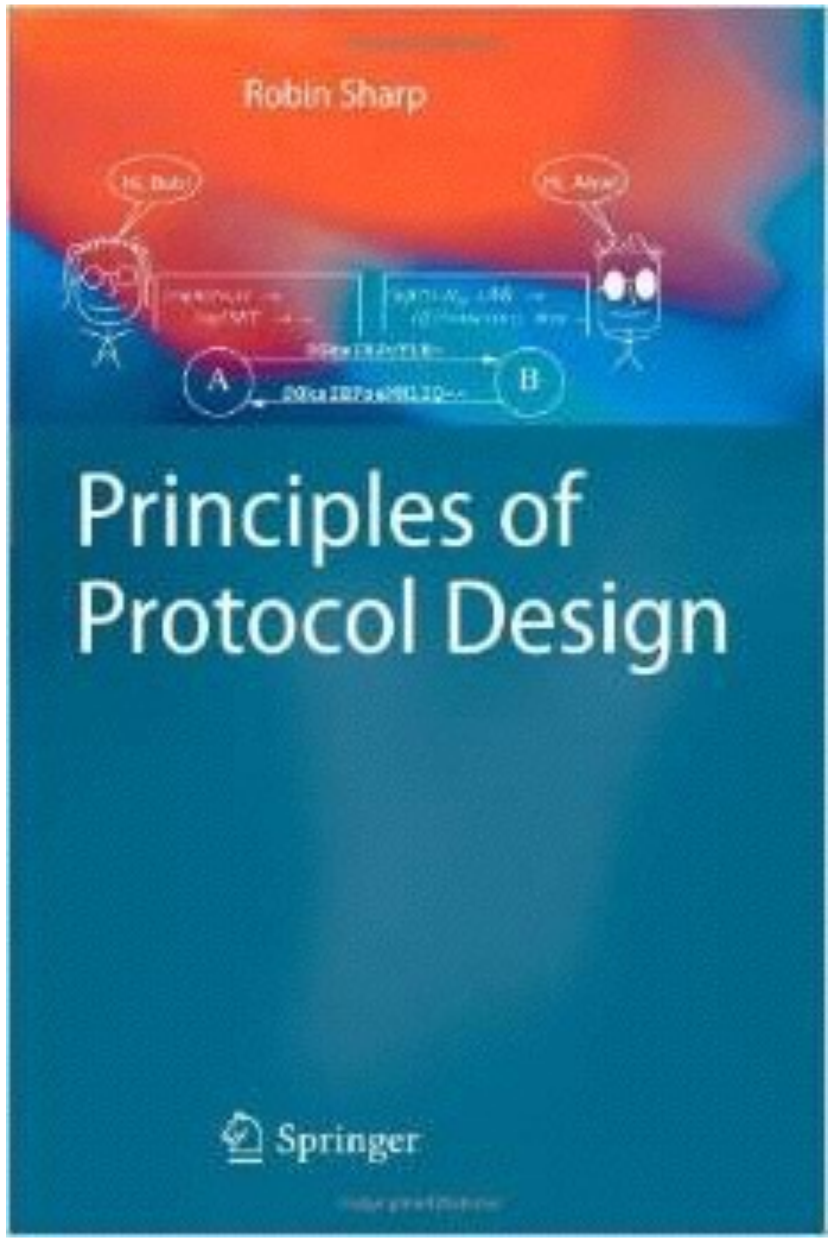


C.A.R.Hoare

Communicating Sequential Processes

PRENTICE-HALL INTERNATIONAL SERIES IN COMPUTER SCIENCE

C.A.R. HOARE SERIES EDITOR



Robin Sharp

Principles of Protocol Design

Springer

Paxos

```
{  Acceptors   }
Proposer     Main       Aux    Learner
|            |  |  |     |       |  -- Phase 2 --
X----------->|->|->|     |       |  Accept!(N,I,V)
|            |  |  !     |       |  --- FAIL! ---
|<-----------X--X--------------->|  Accepted(N,I,V)
|            |  |        |       |  -- Failure detected (only 2 accepted) --
X----------->|->|------->|       |  Accept!(N,I,V)  (re-transmit, include Aux)
|<-----------X--X--------X------>|  Accepted(N,I,V)
|            |  |        |       |  -- Reconfigure : Quorum = 2 --
X----------->|->|        |       |  Accept!(N,I+1,W) (Aux not participating)
|<-----------X--X--------------->|  Accepted(N,I+1,W)
|            |  |        |       |
```

Source: https://en.wikipedia.org/wiki/Paxos_(computer_science)#Byzantine_Paxos

Single
Page
Programmer
Syndrome

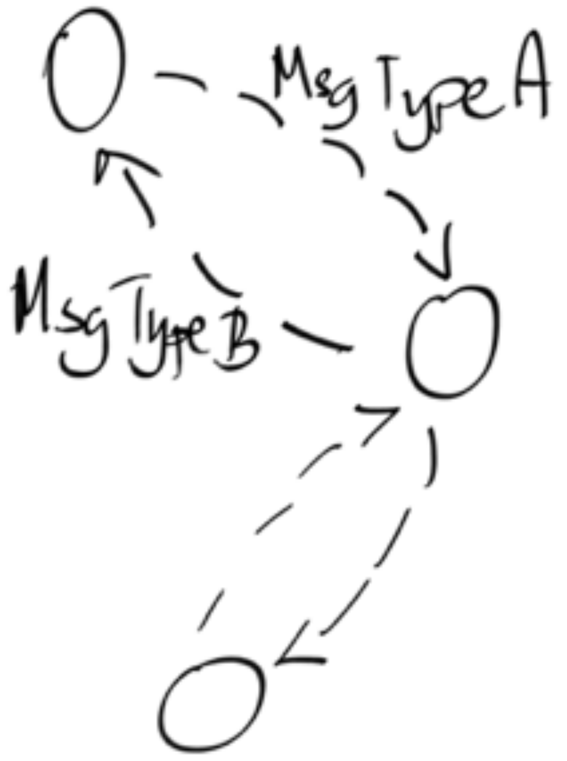
Protocol

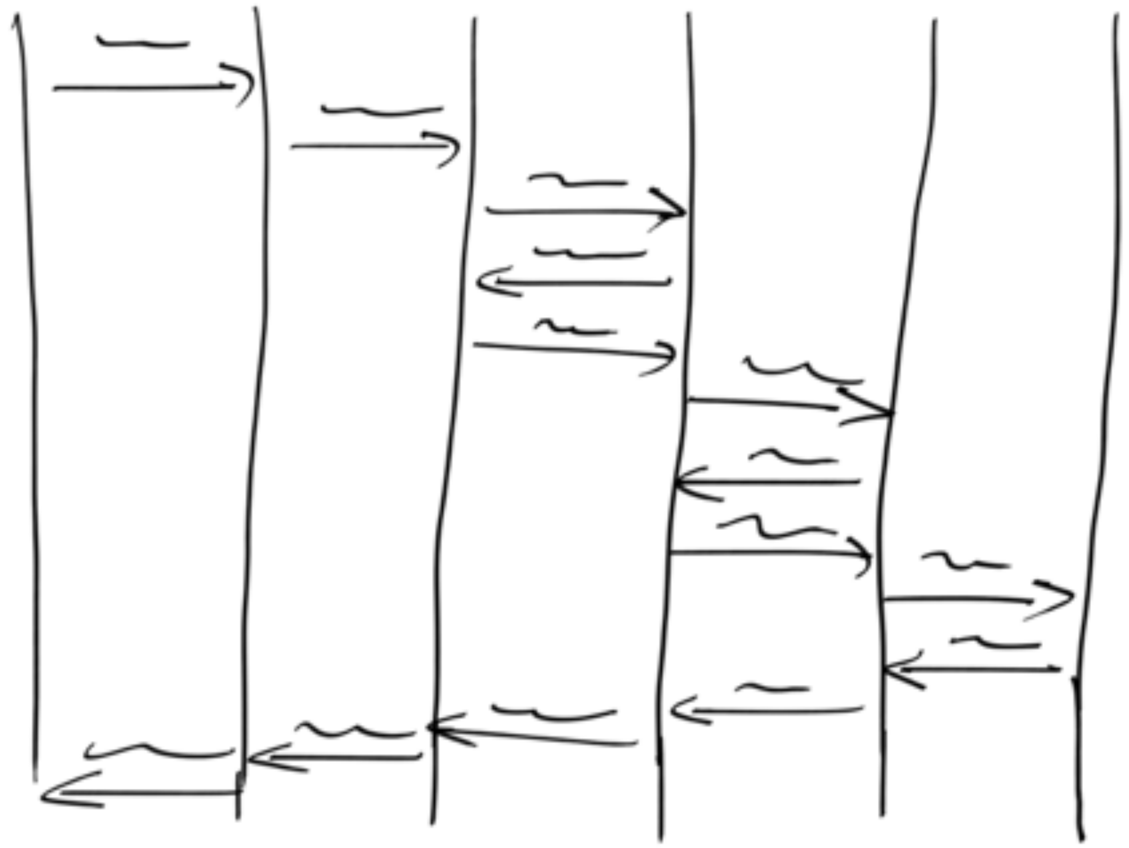
=

How to solve a problem
together

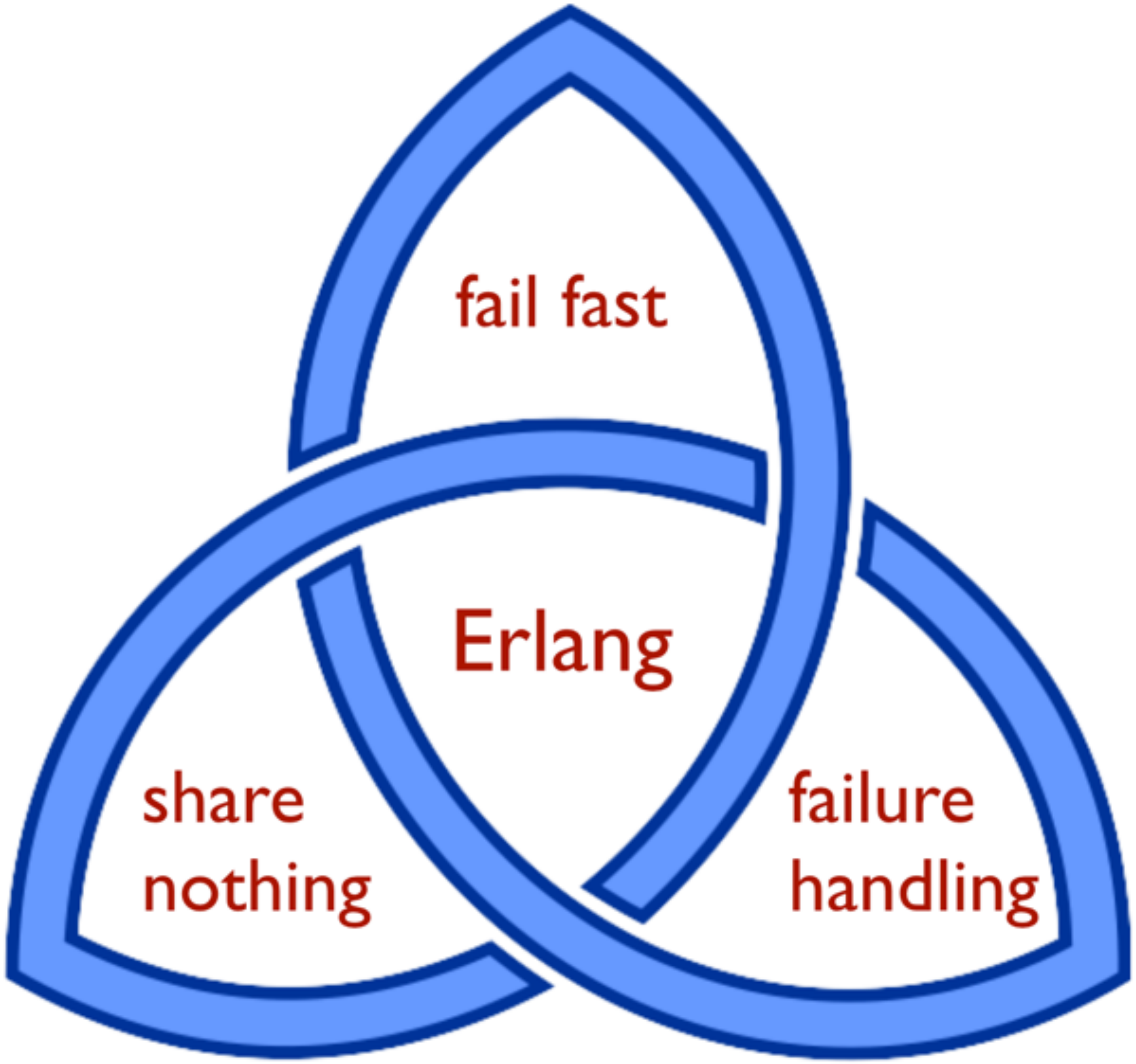
Interaction Diagram

Message Sequence Chart

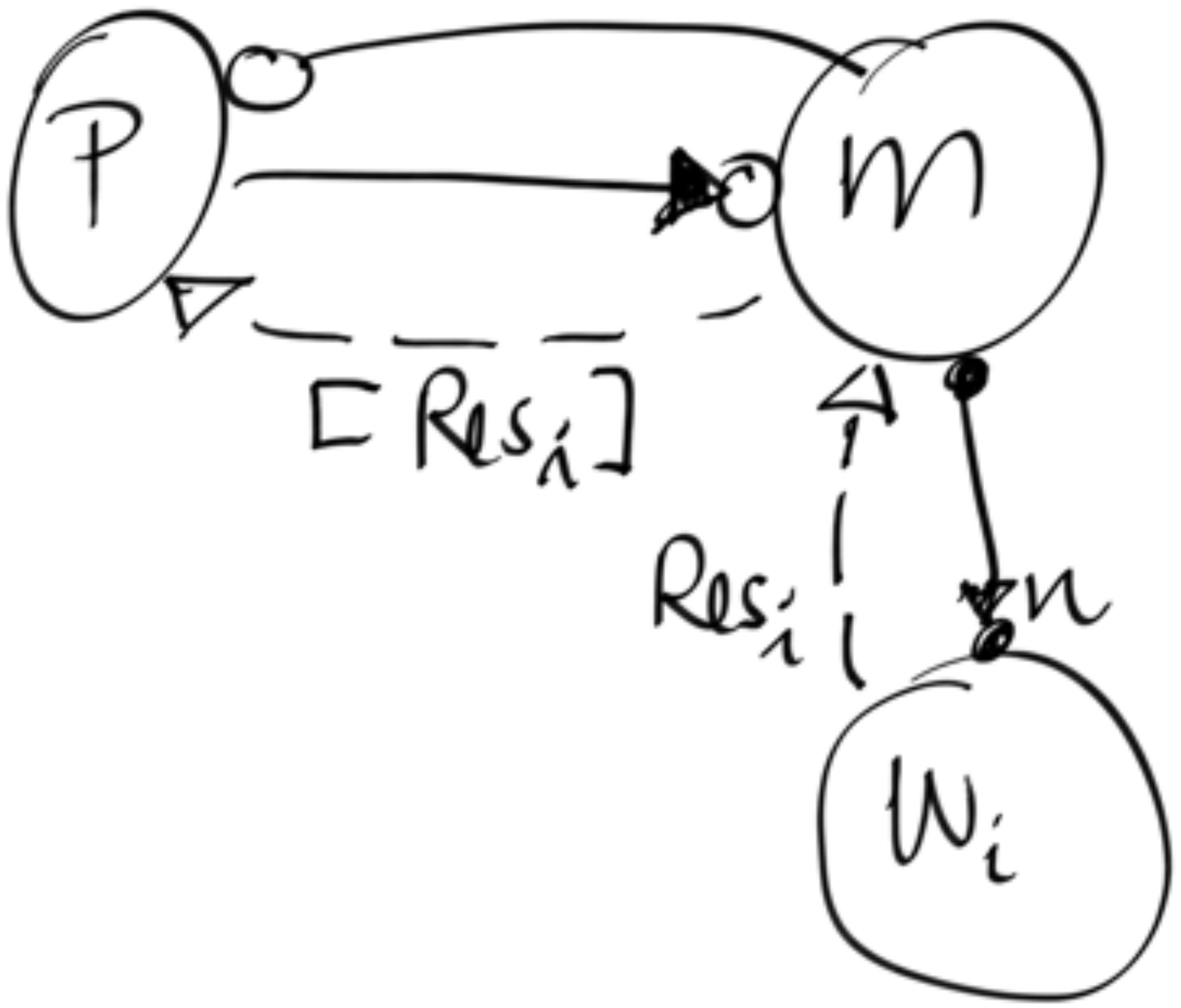




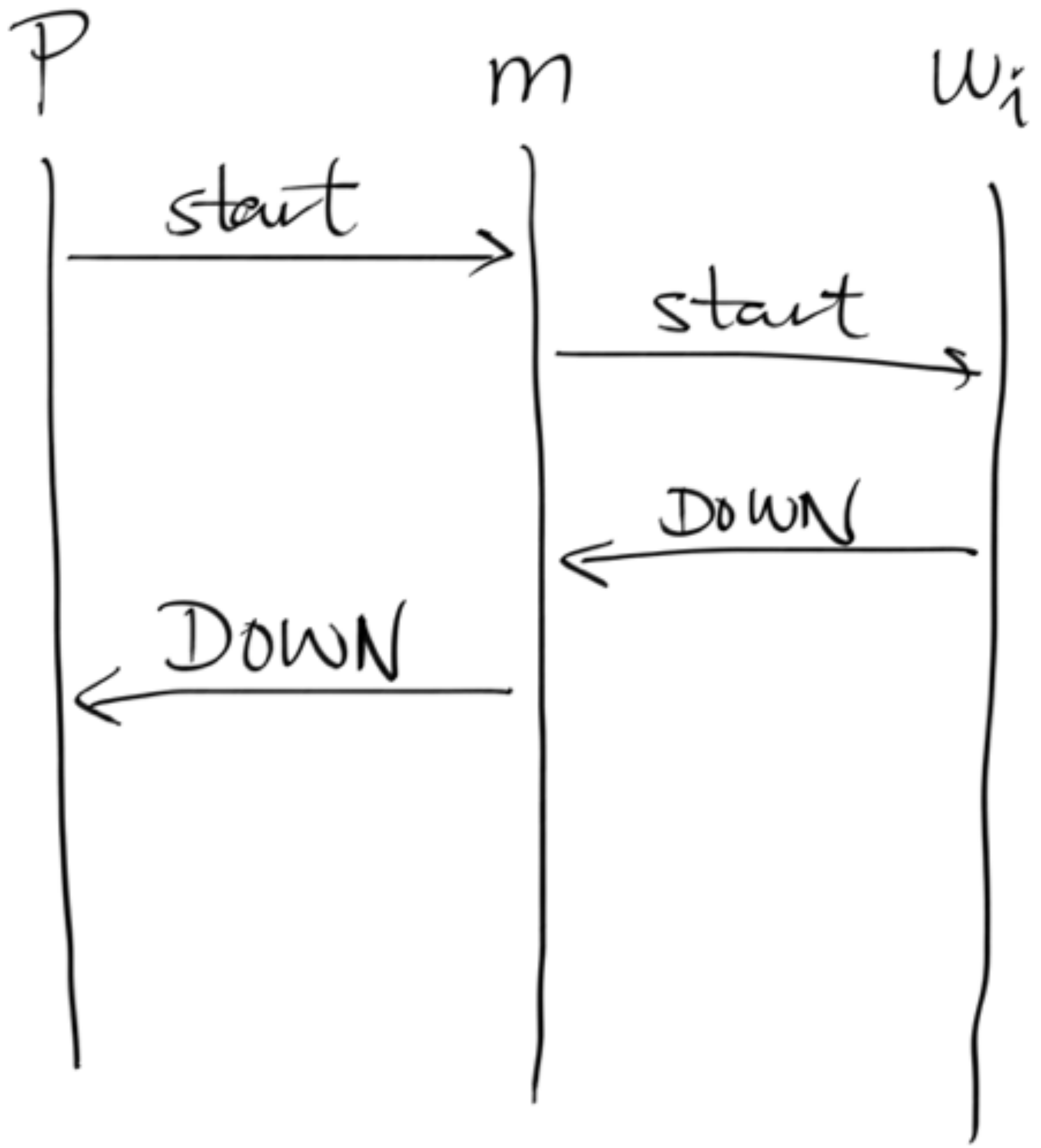
The Golden Trinity Of Erlang



Simple Manager/ Worker Pattern

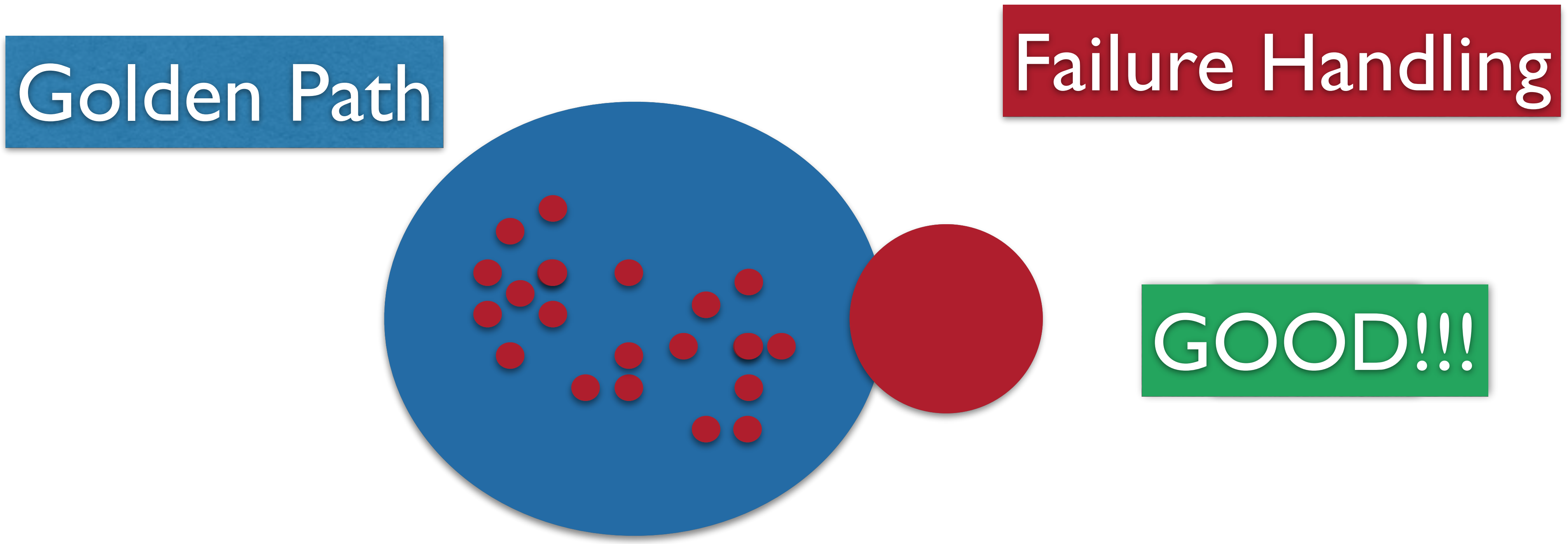


Failures in your protocol



Separation of Concerns

Not embracing failure means you **loose** the ability to handle failures gracefully!



Fault In-Tolerance

Most programming paradigmes are
fault in-tolerant
⇒ must deal with all errors or die

source: http://www.thelmagazine.com/BrooklynAbridged/archives/2013/05/14/should-we-be-worried-about-this-brooklyn-measles-outbreak

Fault Tolerance

Erlang is fault tolerant by design

⇒ failures are embraced and managed

source: http://johnkreng.wordpress.com/tag/jean-claude-van-damme/

Stock Exchange

The Trigger…

Erlang-Questions on using ETS for sell and buy orders:

http://erlang.org/pipermail/erlang-questions/2014-February/077969.html

Painful…

An Exchange

Connects buyers and sellers

Buyers post buy intentions

Sellers post sell intentions

Basic Erlang Idea

One process per buy/sell intention

Processes to negotiate deals
by exchanging messages

Communication

Use gproc as pub-sub mechanism to announce buy and sell intentions

- All buyers listen to sell intention
- All sellers listen to buy intentions

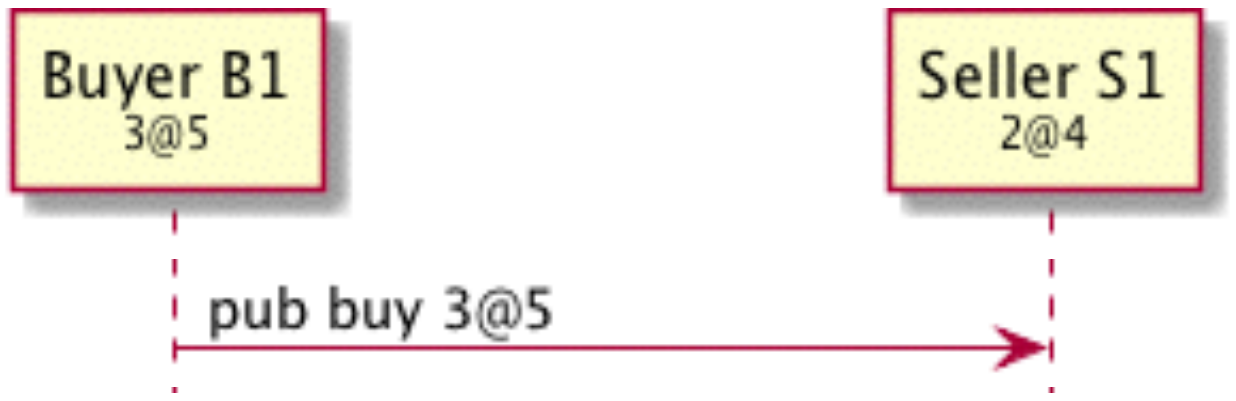
Deals

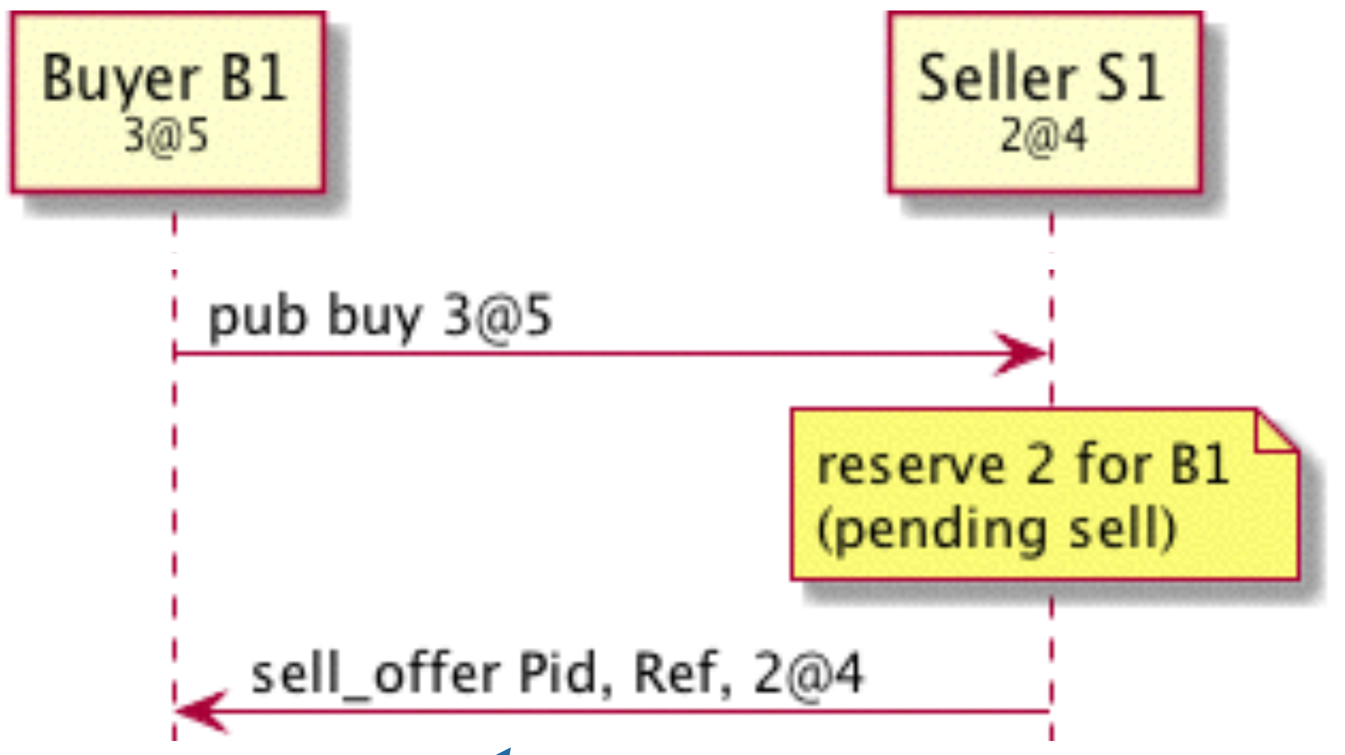
Can happen when

$$price_{seller} \leq price_{buyer}$$

Negotiation by 3-way handshake

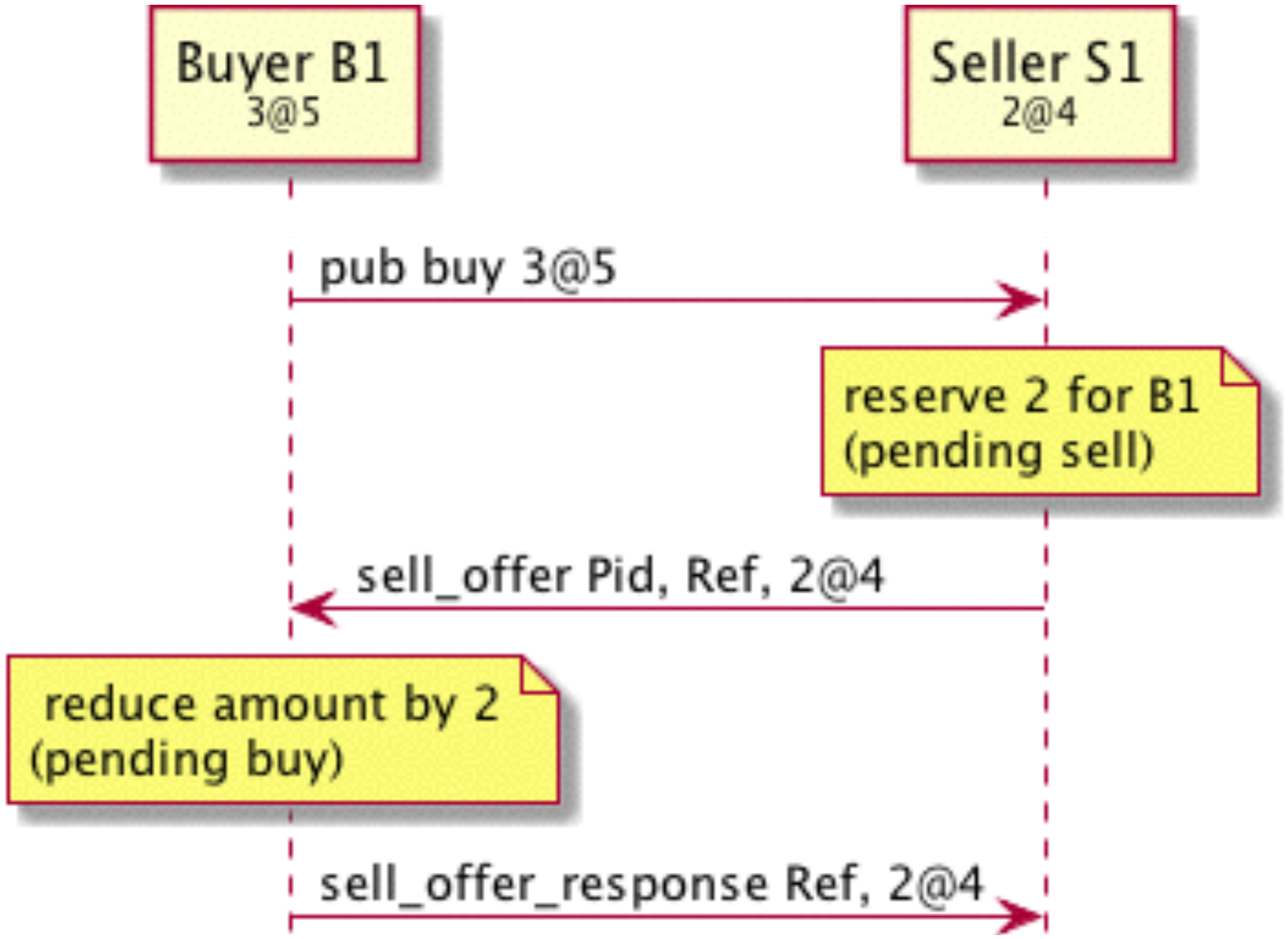
Buyer Arrives

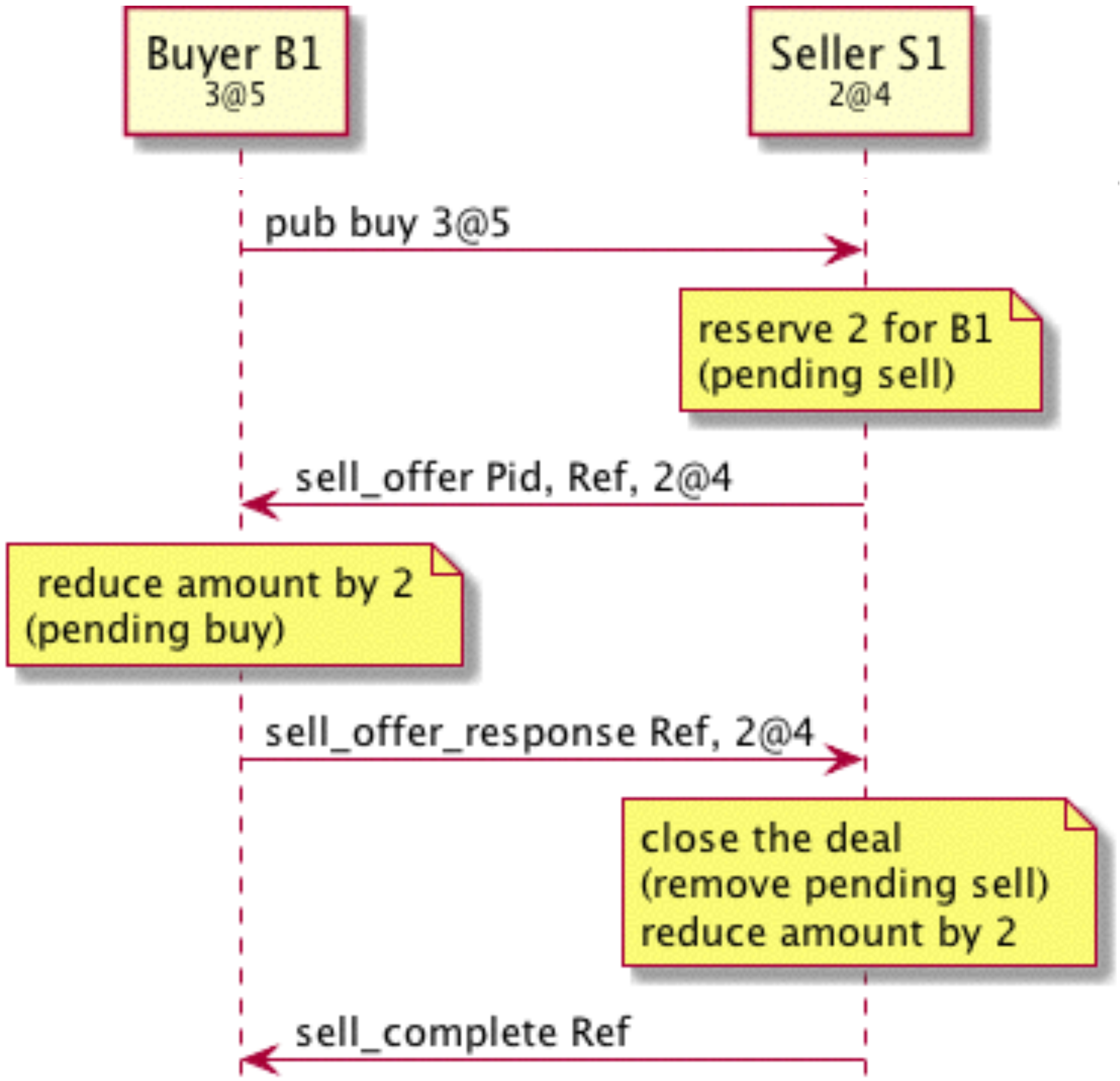

Buyer B1
3@5
Seller S1
2@4
pub buy 3@5

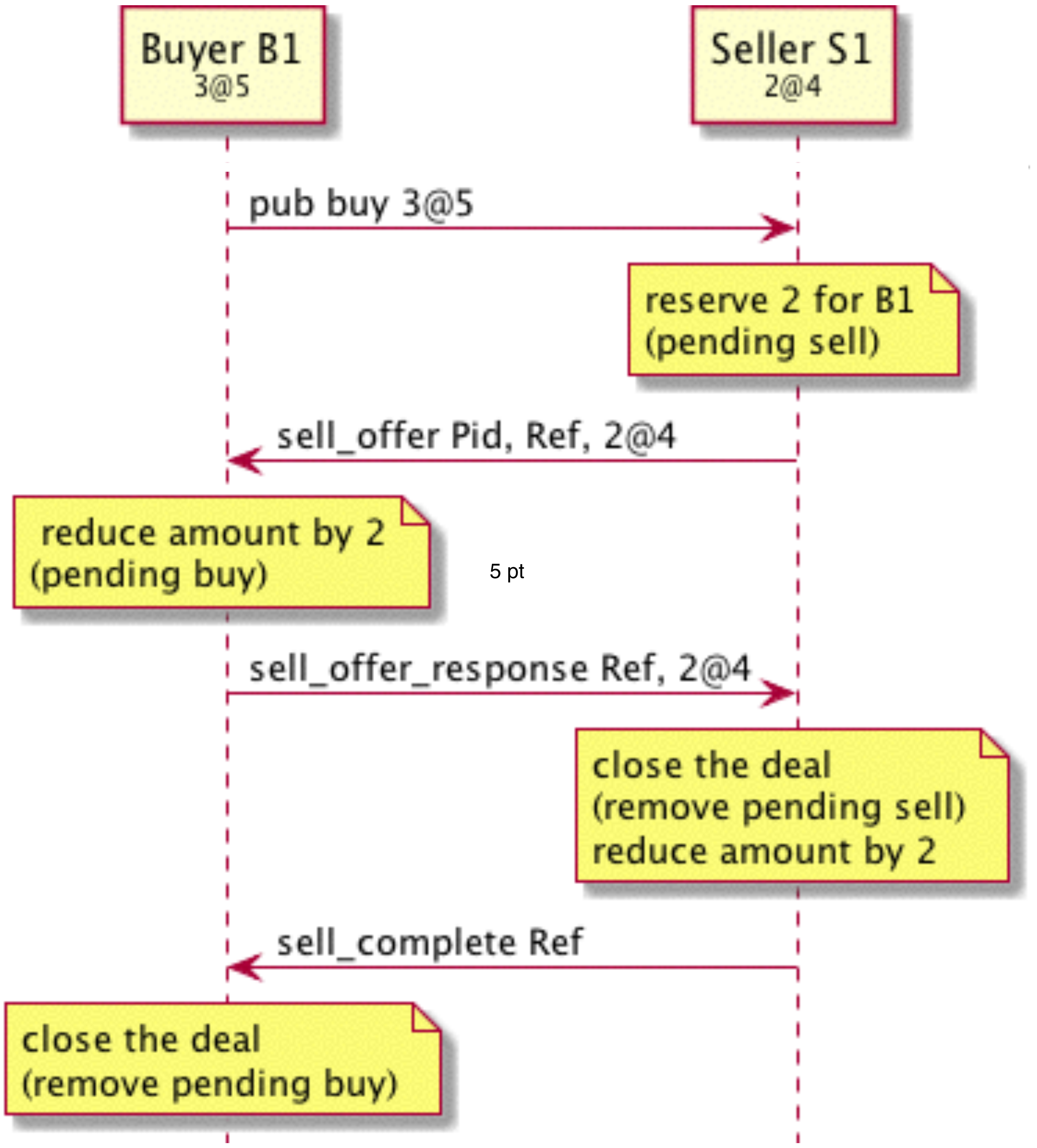


Seller's Pid

Unique reference to identify the sell offer


Buyer B1
3@5
Seller S1
2@4
pub buy 3@5
reserve 2 for B1
(pending sell)
sell_offer Pid, Ref, 2@4
reduce amount by 2
(pending buy)
sell_offer_response Ref, 2@4

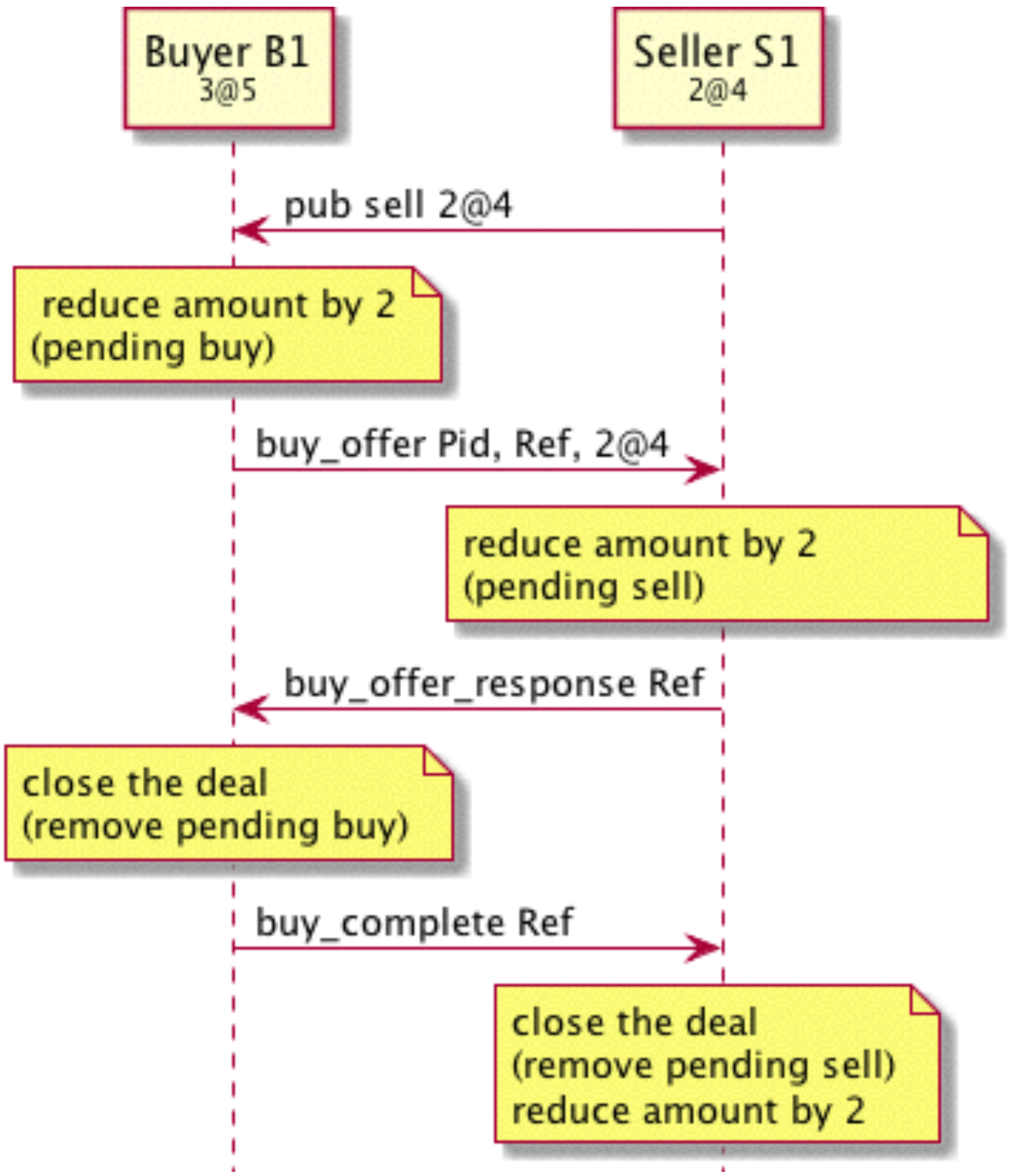

Buyer B1
3@5
Seller S1
2@4
pub buy 3@5
reserve 2 for B1
(pending sell)
sell_offer Pid, Ref, 2@4
reduce amount by 2
(pending buy)
sell_offer_response Ref, 2@4
close the deal
(remove pending sell)
reduce amount by 2
sell_complete Ref



Buyer B1
3@5
Seller S1
2@4
pub buy 3@5
reserve 2 for B1
(pending sell)
sell_offer Pid, Ref, 2@4
reduce amount by 2
(pending buy)
sell_offer_response Ref, 2@4
close the deal
(remove pending sell)
reduce amount by 2
sell_complete Ref
close the deal
(remove pending buy)

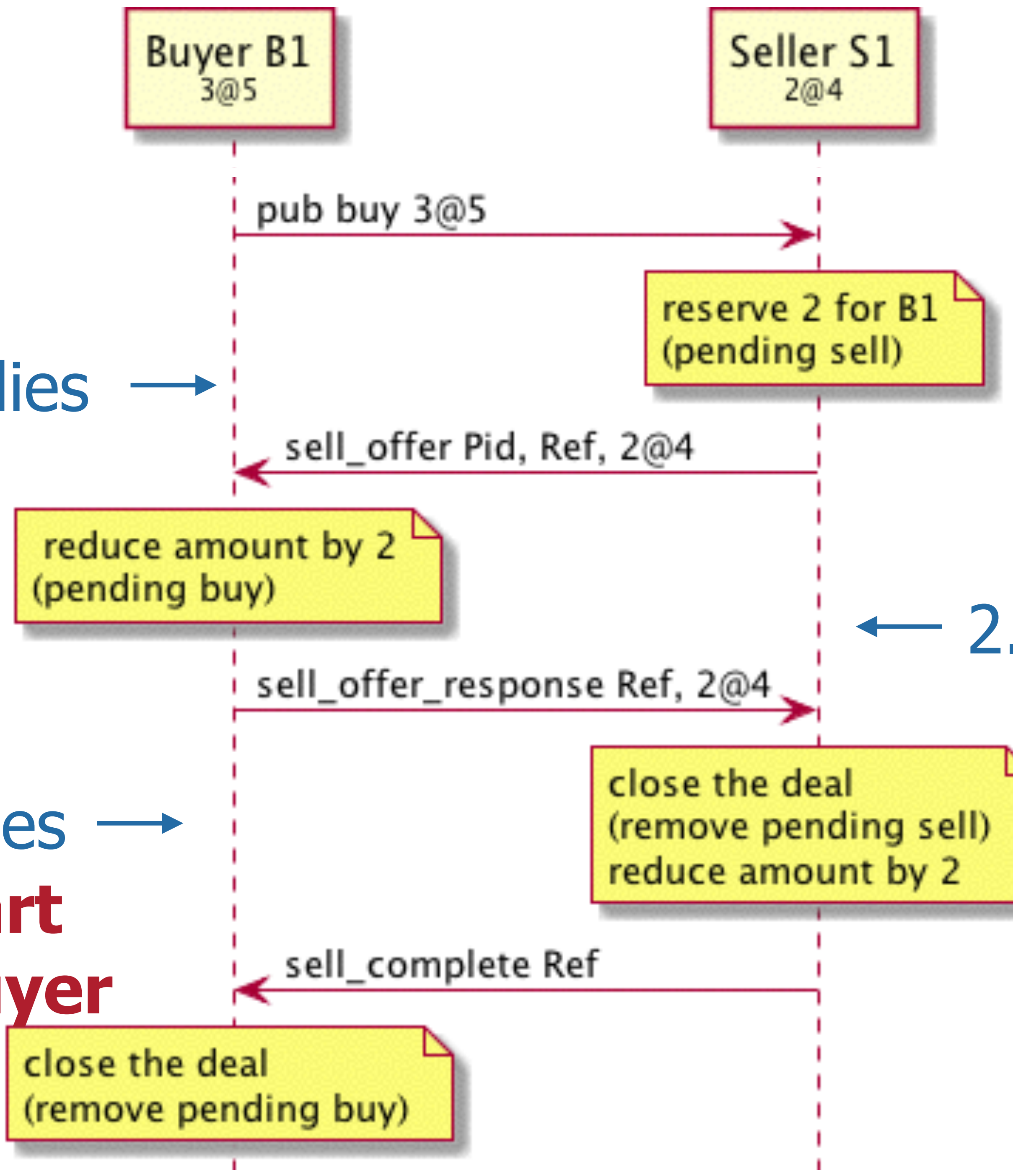
5 pt

Seller Arrives


Buyer B1
3@5
Seller S1
2@4
pub sell 2@4
reduce amount by 2
(pending buy)
buy_offer Pid, Ref, 2@4
reduce amount by 2
(pending sell)
buy_offer_response Ref
close the deal
(remove pending buy)
buy_complete Ref
close the deal
(remove pending sell)
reduce amount by 2

What About Failures?

What Can Go Wrong?



1 & 2 can be fixed by timing out

1. Buyer dies →

← 2. Seller dies

3. Buyer dies →

Simple re-start leaves the buyer at 3@5

Danger!!
Seller has closed the deal on his side

Monitor each other

Removes the need for timeouts

Still not sure how far the other side got

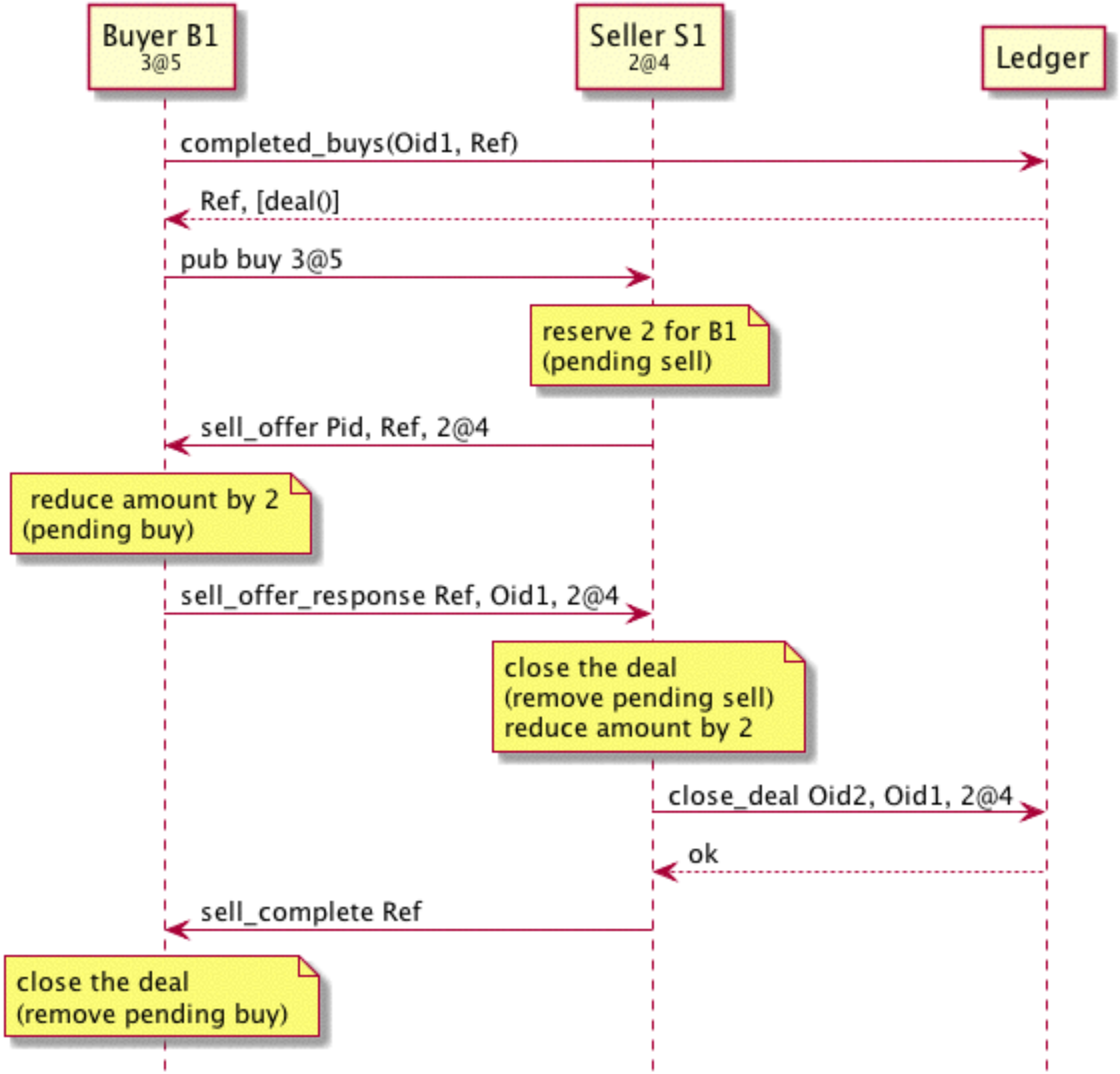
Transaction Log Per Process

- Just replay back to the last state
- Issues:
  - Messages cannot be replayed
  - Must ask partner about their view on the status of the deal

Ledger

Create Ledger process that tracks all completed deals

Each buyer and seller get a unique OfferID when started


Buyer B1
3@5
Seller S1
2@4
Ledger
completed_buys(Oid1, Ref)
Ref, [deal()]
pub buy 3@5
reserve 2 for B1
(pending sell)
sell_offer Pid, Ref, 2@4
reduce amount by 2
(pending buy)
sell_offer_response Ref, Oid1, 2@4
close the deal
(remove pending sell)
reduce amount by 2
close_deal Oid2, Oid1, 2@4
ok
sell_complete Ref
close the deal
(remove pending buy)

Re-cap

A process per cell

Short-lived processes for small tasks

Focus on the protocols between processes

Supervisor to restart

Good Design

Focus on protocols (MSCs)

Ask “What could go wrong here?”

Tools

Lots of processes!!

Supervisors

Link and monitor

Timeouts

Transaction logs (ledgers)

Food for Thought

What can I only do in Erlang?

http://erlang.org/pipermail/erlang-questions/2014-November/081570.html

You can avoid writing your own service framework.

Craig Everett

Testing

Async protocols are nasty

Use EQC - Property Based Testing

- Focus on one process
- Mock the calls to others

Going Deeper

Erlang Matching Business Needs

Thinking Like an Erlanger

Game of life

- https://github.com/lehoff/egol

Erlang Exchange

- https://github.com/lehoff/erlang_exchange

Summary

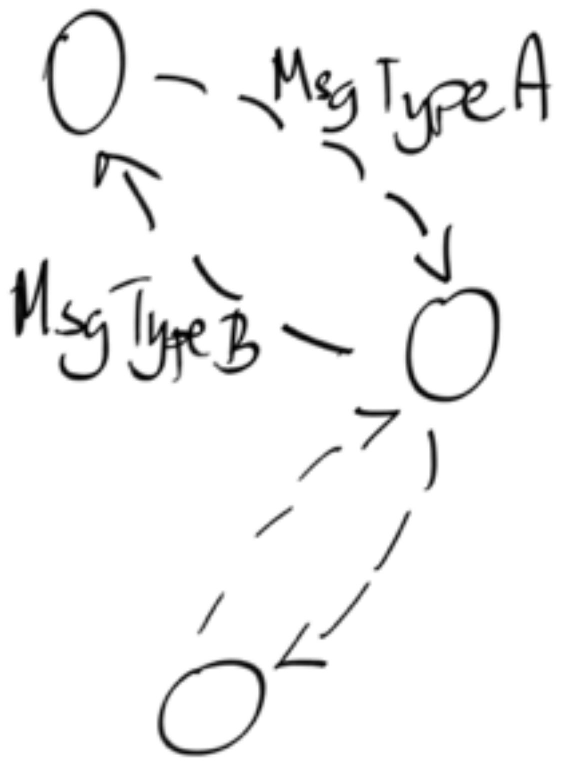
Protocol

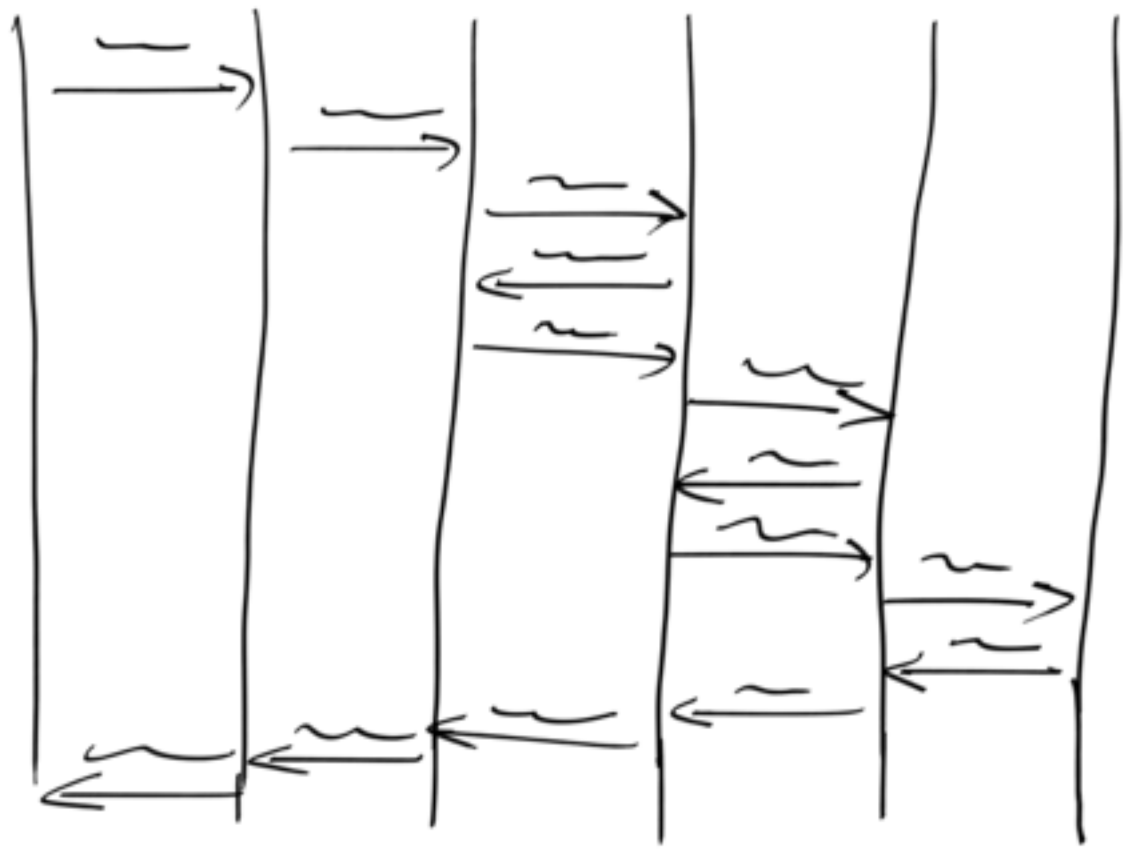
=

How to solve a problem together

Interaction Diagram

Message Sequence Chart





Key building blocks

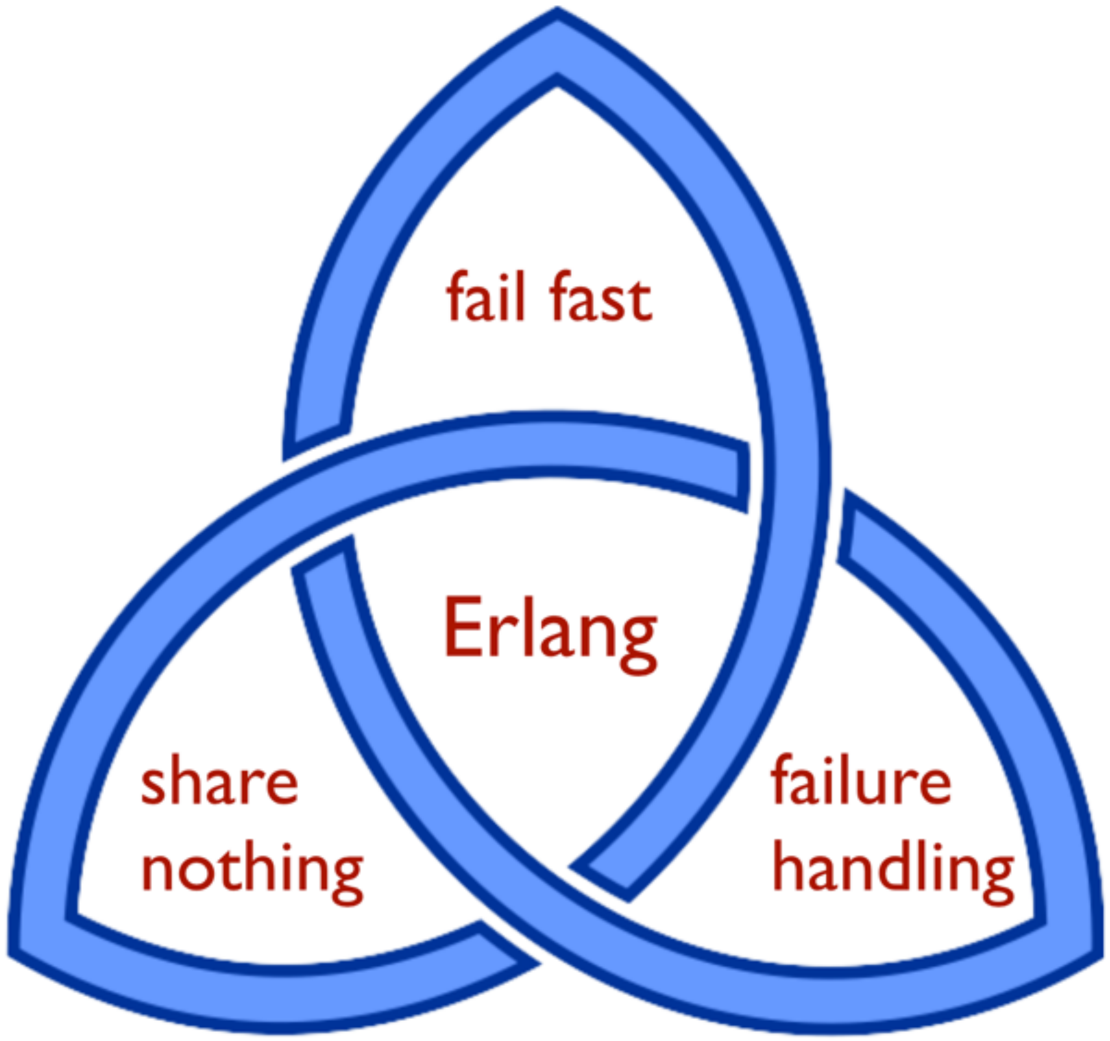
Share nothing processes

Message passing

Fail fast approach

Link/monitor concept

EQC for async testing


fail fast
Erlang
share nothing
failure handling