last time (1)

multi-level page tables

- split virtual page number into parts
- first part: index in 1st level table
- 1st level table points to 2nd level table (instead of data)
- second part: index in 2nd level table
- ...

page table permission bits

- protecting OS memory but making it accessible without changing PTBR
- disabling writes for safe sharing

last time (2)

allocate-on-demand

- don't tell processor about everything "allocated" to program
- fixup disagreement on page fault (instead of crashing program)

copy-on-write

- tell processor: this is read-only
- make a copy and fixup disagreement on protection fault

anonymous feedback (1)

"In OH, some of your TAs are incredibly unhelpful. It is clear that they prioritize their friends over other students who need help. This is not the case for all of them, but when it happens it is incredibly frustration to be waiting and not get help because they are helping in order of who they know and not when students arrived."

anonymous feedback (2)

"the amount of piazza questions for this HW and the overwhelming amount of people at OH indicates that the topic was not taught well and the instructions are unclear"

selected common confusion on assignment

translate ≈ processor's memory lookup

- can follow lookup diagram from slides (1- or 2-level)

page_allocate ≈ OS's allocation-on-demand

- allocate things that translate would find missing
- allocates both page tables and the 'data' they point to
  - assignment originally wasn't explicit about this, but translate() can't work if you don't
- needs to handle initializing page tables (so translate knows there's nothing there yet)

how big are virtual addresses (the part used for translation)?

- based on page table sizes
- not all of size_t used

2-level example

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused

page table base register `0x20`; translate virtual address `0x131`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	00 91 72 13
0x24-7	D4 F5 36 07
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	DB 0B DB 0B
0x38-B	EC 0C EC 0C
0x3C-F	FC 0C FC 0C

2-level example

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused

page table base register `0x20`; translate virtual address `0x131`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	00 91 72 13
0x24-7	D4 F5 36 07
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	DB 0B DB 0B
0x38-B	EC 0C EC 0C
0x3C-F	FC 0C FC 0C

0x131 = 1 0011 0001

0x20 + 0x4 × 1 = 0x24

PTE 1 value:

0xD4 = 1101 0100

PPN 110, valid 1

2-level example

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused

page table base register `0x20`; translate virtual address `0x131`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	00 91 72 13
0x24-7	D4 F5 36 07
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	DB 0B DB 0B
0x38-B	EC 0C EC 0C
0x3C-F	FC 0C FC 0C

0x131 = 1 0011 0001

0x20 + 0x4 × 1 = 0x24

PTE 1 value:

0xD4 = 1101 0100

PPN 110, valid 1

PTE 2 addr:

110 000 + 110 × 1 = 0x36

PTE 2 value: 0xDB

2-level example

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused

page table base register `0x20`; translate virtual address `0x131`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	00 91 72 13
0x24-7	D4 F5 36 07
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	DB 0B DB 0B
0x38-B	EC 0C EC 0C
0x3C-F	FC 0C FC 0C

$\texttt{0x131} = \texttt{1 0011 0001}$

$\texttt{0x20} + \texttt{0x4} \times 1 = \texttt{0x24}$

PTE 1 value:

$\texttt{0xD4} = \texttt{1101 0100}$

PPN `110`, valid 1

PTE 2 addr:

$\texttt{110 000} + \texttt{110} \times 1 = \texttt{0x36}$

PTE 2 value: `0xDB`

PPN `110`; valid 1

M[`110 001` (`0x31`)] = `0x0A`

2-level example

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused

page table base register `0x20`; translate virtual address `0x131`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	00 91 72 13
0x24-7	D4 F5 36 07
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	DB 0B DB 0B
0x38-B	EC 0C EC 0C
0x3C-F	FC 0C FC 0C

0x131 = 1 0011 0001

0x20 + 0x4 ×1 = 0x24

PTE 1 value:

0xD4 = 1101 0100

PPN 110, valid 1

PTE 2 addr:

110 000 + 110 × 1 = 0x36

PTE 2 value: 0xDB

PPN 110; valid 1

M[110 001 (0x31)] = 0x0A

2-level example

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused

page table base register `0x20`; translate virtual address `0x131`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 3A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	00 91 72 13
0x24-7	D4 F5 36 07
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	DB 0B DB 0B
0x38-B	EC 0C EC 0C
0x3C-F	FC 0C FC 0C

0x131 = 1 0011 0001

0x20 + 0x4 ×1 = 0x24

PTE 1 value:

0xD4 = 1101 0100

PPN 110, valid 1

PTE 2 addr:

110 000 + 110 × 1 = 0x36

PTE 2 value: 0xDB

PPN 110; valid 1

M[110 001 (0x31)] = 0x0A

2-level splitting

9-bit virtual address

6-bit physical address

8-byte pages $\rightarrow$ 3-bit page offset (bottom bits)

9-bit VA: 6 bit VPN + 3 bit PO

6-bit PA: 3 bit PPN + 3 bit PO

8 entry page tables $\rightarrow$ 3-bit VPN parts

9-bit VA: 3 bit VPN part 1; 3 bit VPN part 2

2-level exercise (2)

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused;

page table base register `0x10`; translate virtual address `0x109`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 5A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	D4 D5 D6 D7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	DB 0B DB 0B
0x38-B	EC 0C EC 0C
0x3C-F	FC 0C FC 0C

2-level exercise (2)

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused;

page table base register `0x10`; translate virtual address `0x109`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 5A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	D4 D5 D6 D7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	DB 0B DB 0B
0x38-B	EC 0C EC 0C
0x3C-F	FC 0C FC 0C

`0x109` = `100 011 001`

(PTE 1 at:

0x10 + PTE size times 4 (100))

PTE 1: `0x1B` at `0x14`

PTE 1: PPN `000` (0) valid 1

(second table at:

0 (000) times page size = 0x00)

PTE 2: `0x33` at `0x03`

PTE 2: PPN `001` (1) valid 1

`001 001` = `0x09` → `0x99`

2-level exercise (2)

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused;

page table base register `0x10`; translate virtual address `0x109`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 5A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	D4 D5 D6 D7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	DB 0B DB 0B
0x38-B	EC 0C EC 0C
0x3C-F	FC 0C FC 0C

`0x109` = `100 011 001`

(PTE 1 at:

0x10 + PTE size times 4 (100))

PTE 1: `0x1B` at `0x14`

PTE 1: PPN `000` (0) valid 1

(second table at:

0 (000) times page size = 0x00)

PTE 2: `0x33` at `0x03`

PTE 2: PPN `001` (1) valid 1

`001 001` = `0x09` → `0x99`

2-level exercise (2)

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused;

page table base register `0x10`; translate virtual address `0x109`

physical addresses	bytes	physical addresses	bytes
0x00-3	00 11 22 33	0x20-3	D0 D1 D2 D3
0x04-7	44 55 66 77	0x24-7	D4 D5 D6 D7
0x08-B	88 99 AA BB	0x28-B	89 9A AB BC
0x0C-F	CC DD EE FF	0x2C-F	CD DE EF F0
0x10-3	1A 2A 5A 4A	0x30-3	BA 0A BA 0A
0x14-7	1B 2B 3B 4B	0x34-7	DB 0B DB 0B
0x18-B	1C 2C 3C 4C	0x38-B	EC 0C EC 0C
0x1C-F	1C 2C 3C 4C	0x3C-F	FC 0C FC 0C

`0x109` = `100 011 001`

(PTE 1 at:

0x10 + PTE size times 4 (100))

PTE 1: `0x1B` at `0x14`

PTE 1: PPN `000` (0) valid 1

(second table at:

0 (000) times page size = 0x00)

PTE 2: `0x33` at `0x03`

PTE 2: PPN `001` (1) valid 1

`001 001` = `0x09` → `0x99`

2-level exercise (2)

9-bit virtual addresses, 6-bit physical; 8 byte pages, 1 byte PTE

page tables 1 page; PTE: 3 bit PPN (MSB), 1 valid bit, 4 unused;

page table base register `0x10`; translate virtual address `0x109`

physical addresses	bytes
0x00-3	00 11 22 33
0x04-7	44 55 66 77
0x08-B	88 99 AA BB
0x0C-F	CC DD EE FF
0x10-3	1A 2A 5A 4A
0x14-7	1B 2B 3B 4B
0x18-B	1C 2C 3C 4C
0x1C-F	1C 2C 3C 4C

physical addresses	bytes
0x20-3	D0 D1 D2 D3
0x24-7	D4 D5 D6 D7
0x28-B	89 9A AB BC
0x2C-F	CD DE EF F0
0x30-3	BA 0A BA 0A
0x34-7	DB 0B DB 0B
0x38-B	EC 0C EC 0C
0x3C-F	FC 0C FC 0C

`0x109` = `100 011 001`

(PTE 1 at:

0x10 + PTE size times 4 (100))

PTE 1: `0x1B` at `0x14`

PTE 1: PPN `000` (0) valid 1

(second table at:

0 (000) times page size = 0x00)

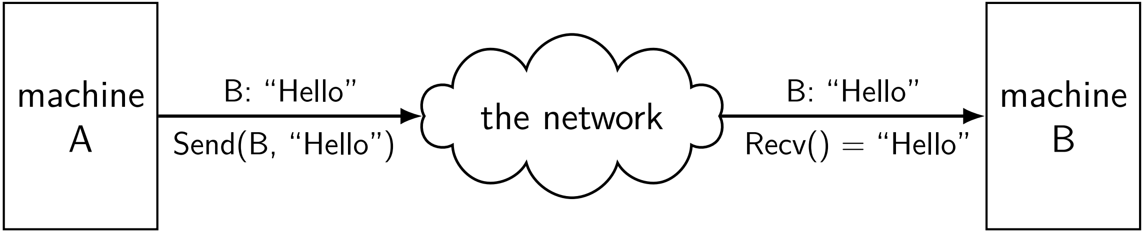
PTE 2: `0x33` at `0x03`

PTE 2: PPN `001` (1) valid 1

`001 001` = `0x09` → `0x99`

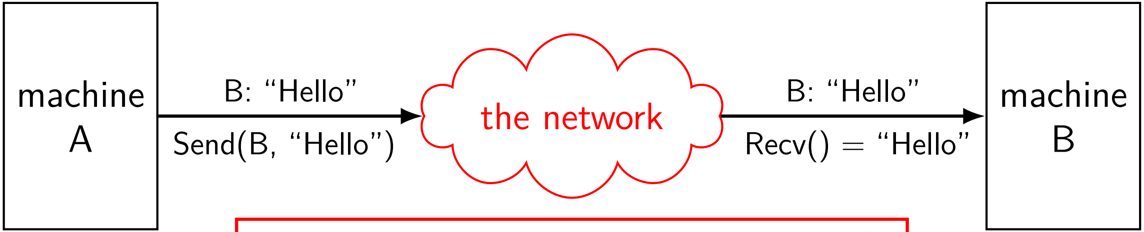
mailbox model

mailbox abstraction: send/receive messages



mailbox model

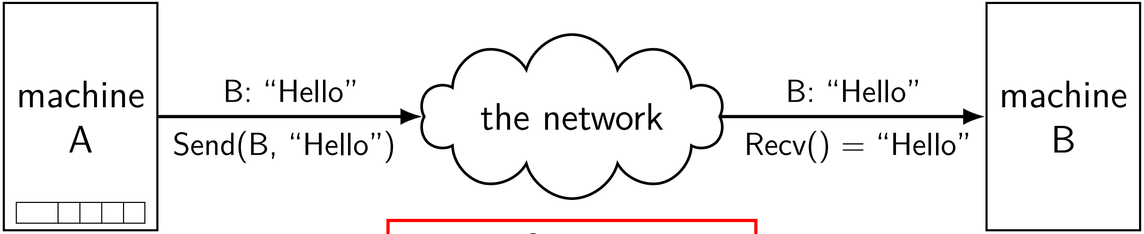
mailbox abstraction: send/receive messages



network knows how to get message to B

mailbox model

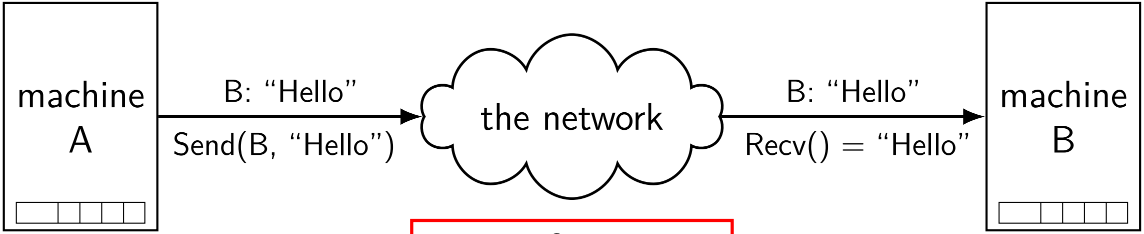
mailbox abstraction: send/receive messages



queue of messages
from sending program
waiting to be sent

mailbox model

mailbox abstraction: send/receive messages



queue of messages not yet received by receiving program

connections over mailboxes

real Internet: mailbox-style communication

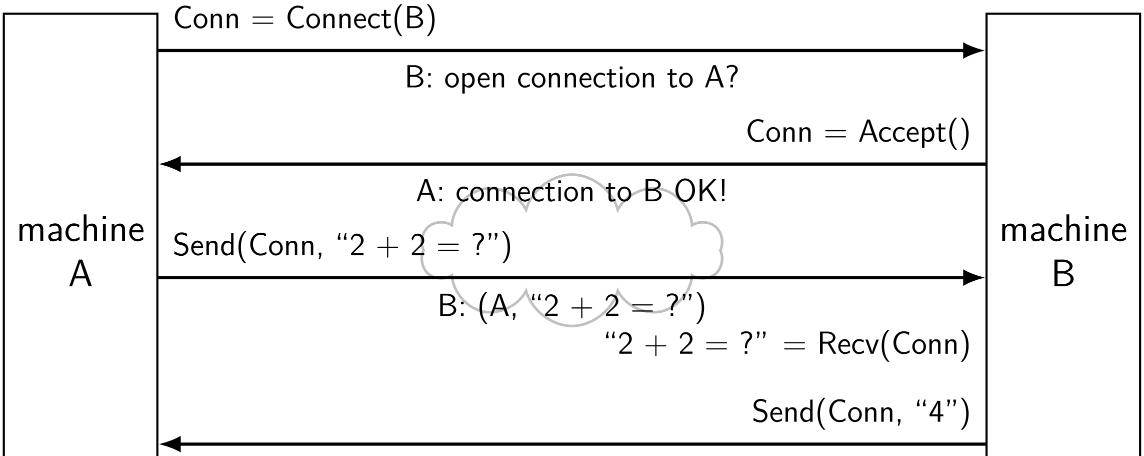
- send packets to particular mailboxes
- no gaurentee on order, when received

sockets implemented on top of this

conections

connections: two-way channel for messages

extra operations: connect, accept



recall: sockets

open connection then …

read+write just like a terminal file

doesn't look like individual messages

"connection abstraction"

layers

application	HTTP, SSH, SMTP, ...	application-defined meanings
transport	TCP, UDP, ...	reach correct program, reliablity/streams
network	IPv4, IPv6, ...	reach correct machine (across networks)
link	Ethernet, Wi-Fi, ...	coordinate shared wire/radio
physical	...	encode bits for wire/radio

layers

application	HTTP, SSH, SMTP, …	application-defined meanings
transport	TCP, UDP, …	reach correct program, reliablity/streams
network	IPv4, IPv6, …	reach correct machine (across networks)
link	Ethernet, Wi-Fi, …	coordinate shared wire/radio
physical	…	encode bits for wire/radio

network limitations/failures

messages lost

messages delayed/reordered

messages limited in size

messages corrupted

network limitations/failures

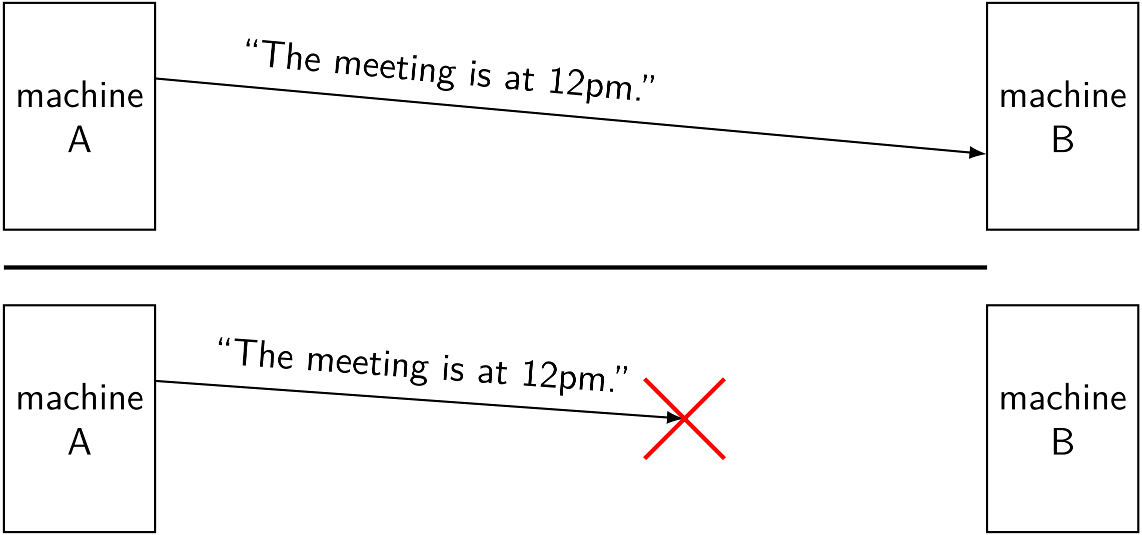
messages lost

messages delayed/reordered

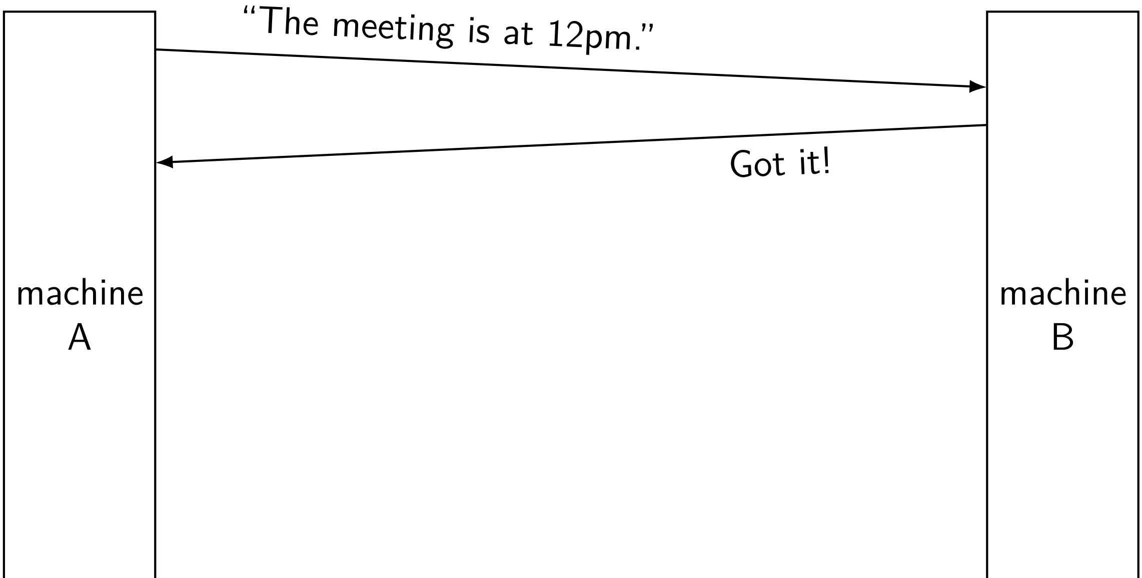
messages limited in size

messages corrupted

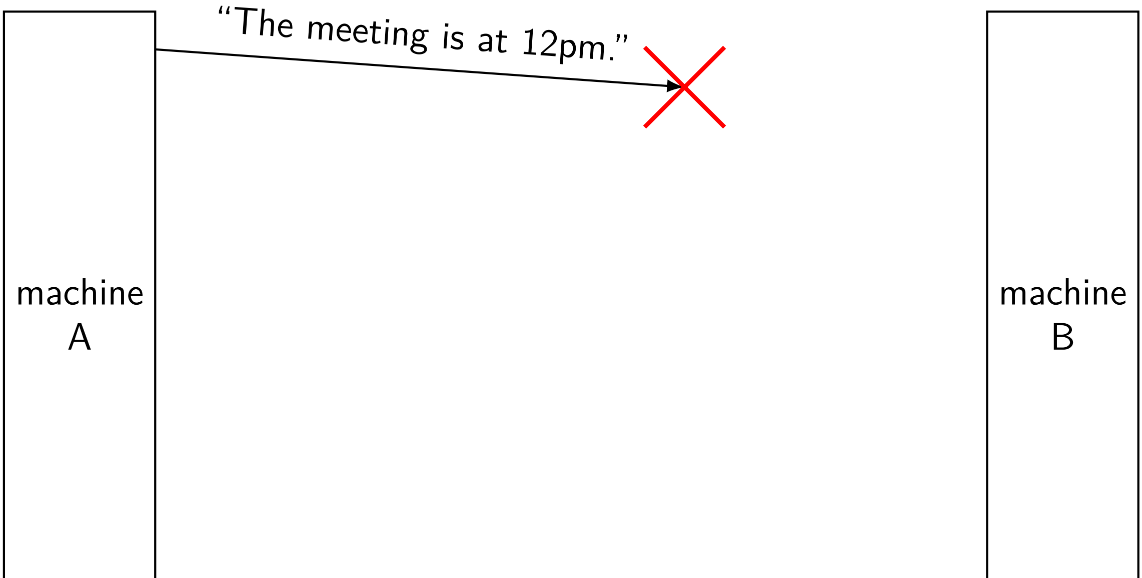
dealing with network message lost



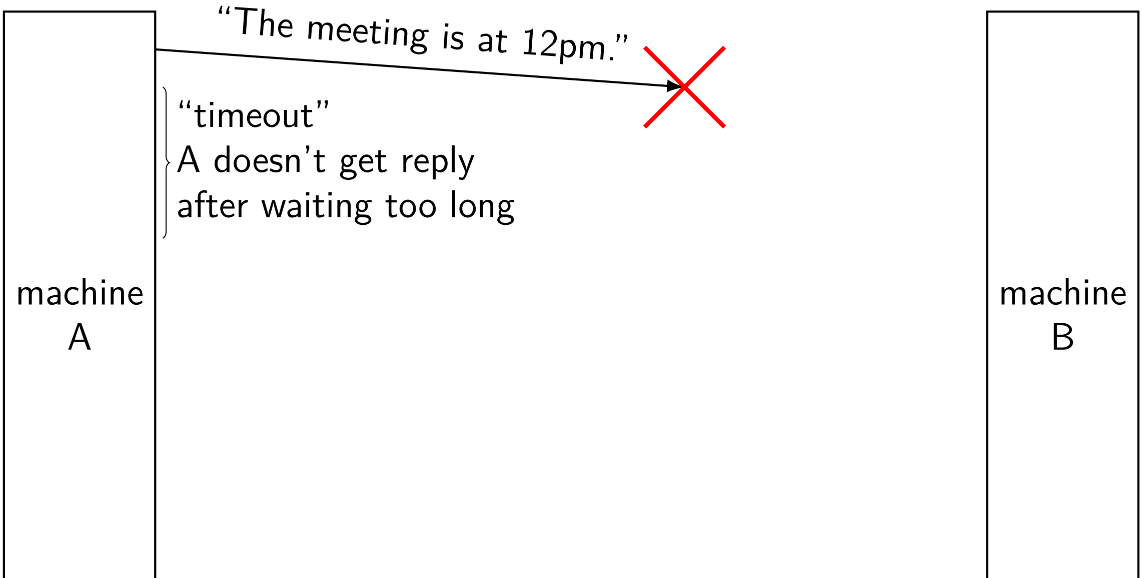
handling lost message: acknowledgements



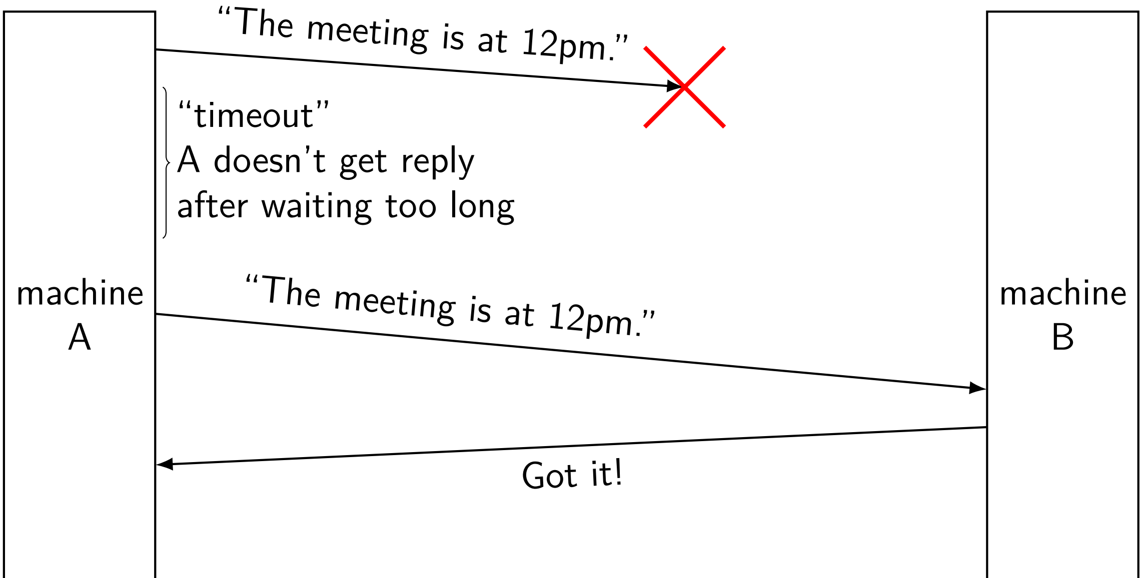
handling lost message



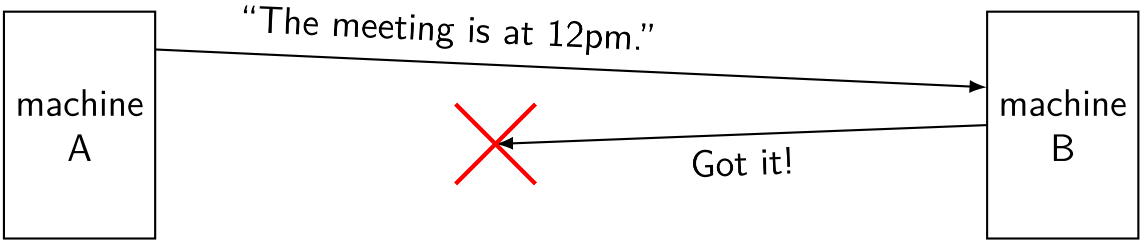
handling lost message



handling lost message



exercise: lost acknowledgement



exercise: how to fix this?

A. machine A needs to send "Got 'got it!' "
B. machine B should resend "Got it!" on its own
C. machine A should resend the original message on its own
D. none of these

answers

send “Got ‘got it!’ ”?

- same problem: Now send ‘Got Got Got it’?

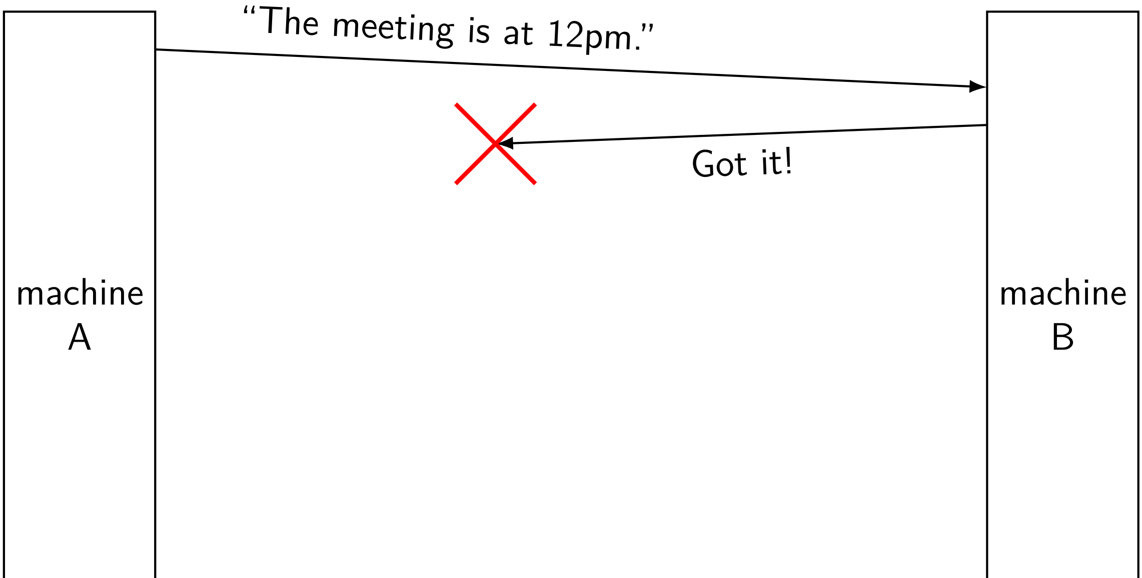
resend “Got it!” own its own?

- how many times? — B doesn’t have that info

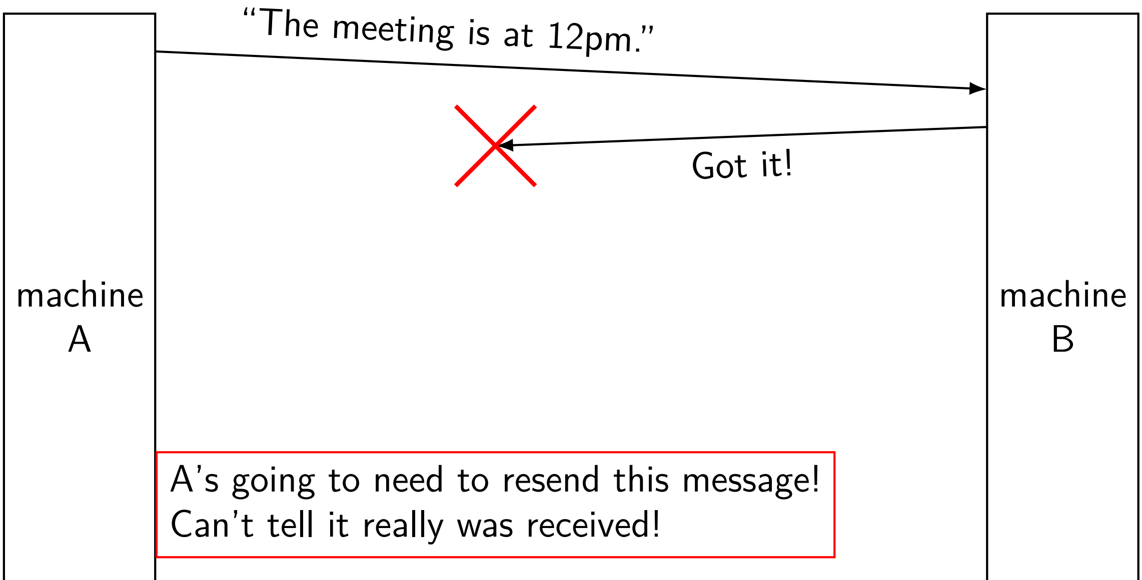
resend original message?

- yes!
- as far as machine A can be, *exact same situation* as losing original message

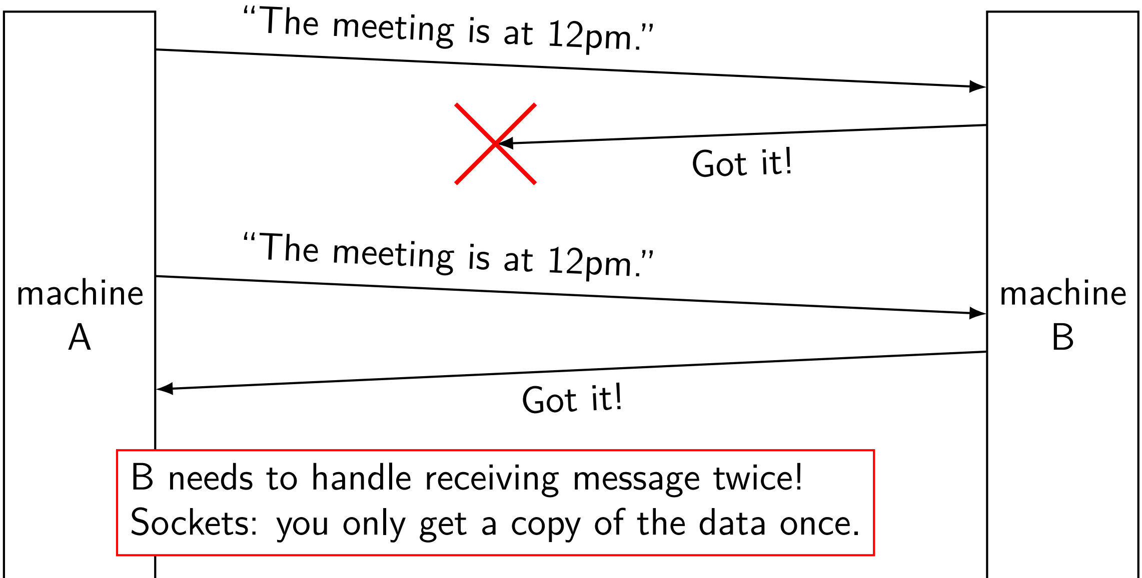
lost acknowledgements



lost acknowledgements



lost acknowledgements



B needs to handle receiving message twice!
Sockets: you only get a copy of the data once.

network limitations/failures

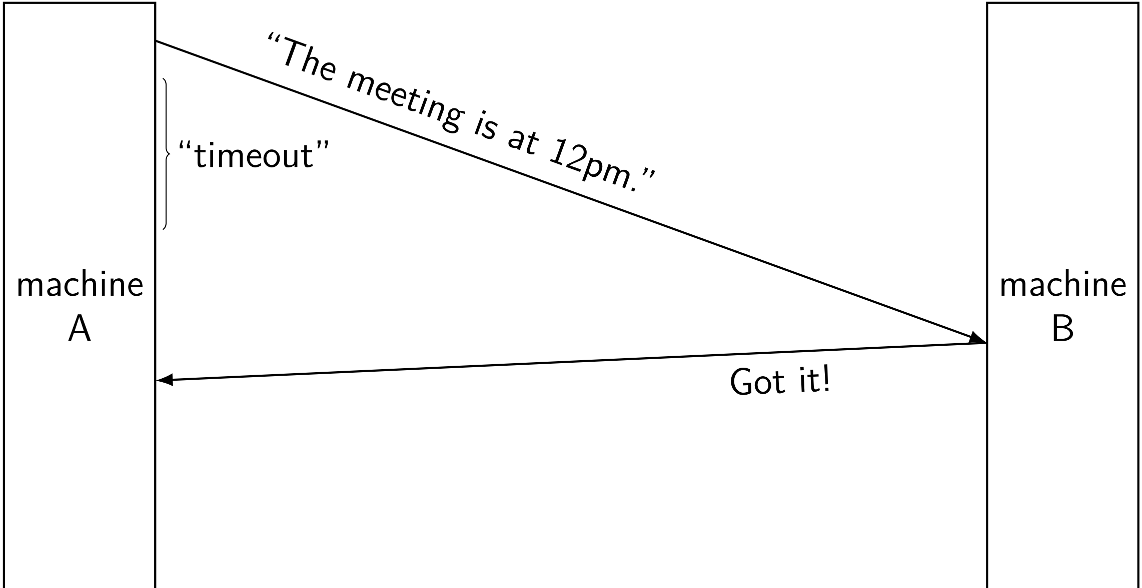
messages lost

messages delayed/reordered

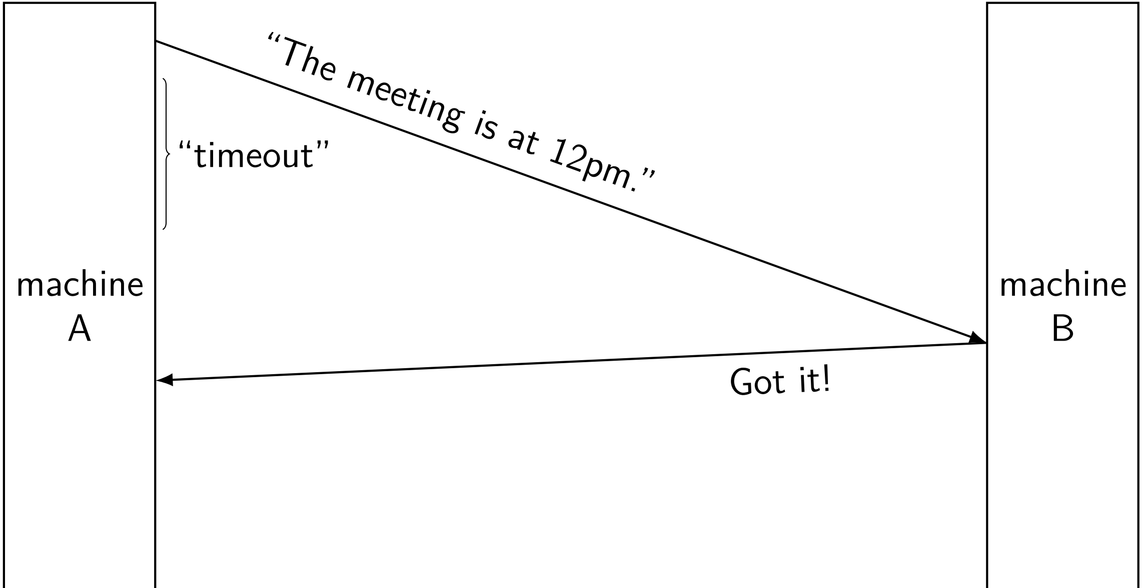
messages limited in size

messages corrupted

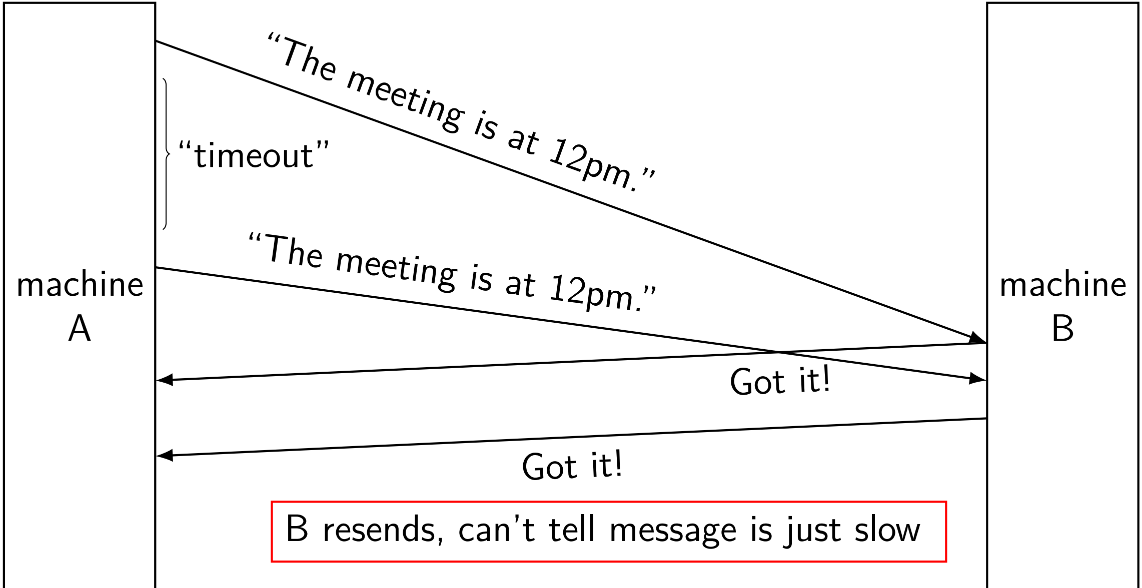
delayed message



delayed message

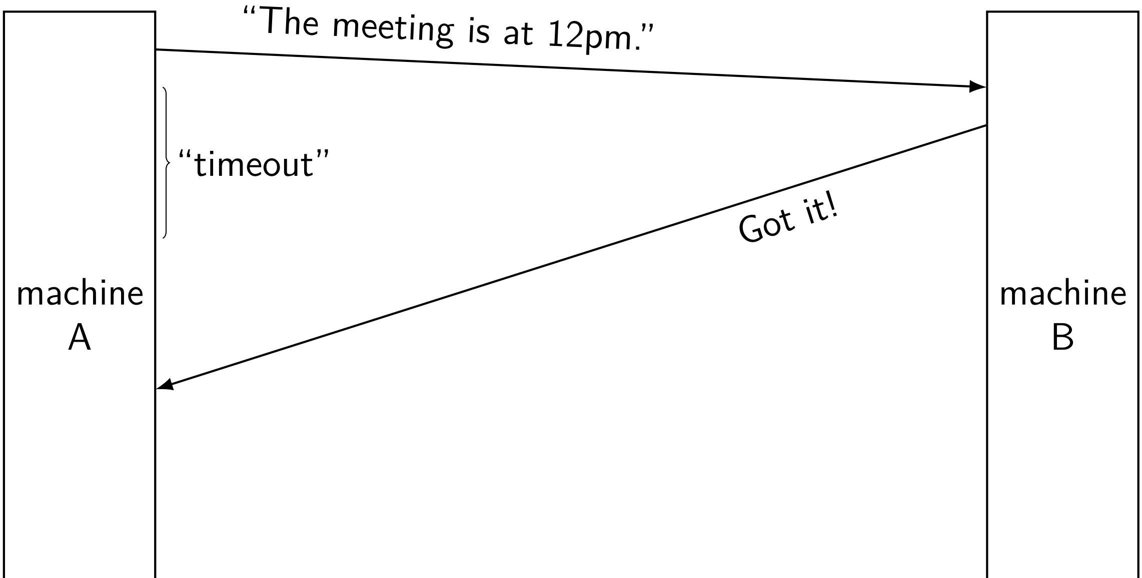


delayed message

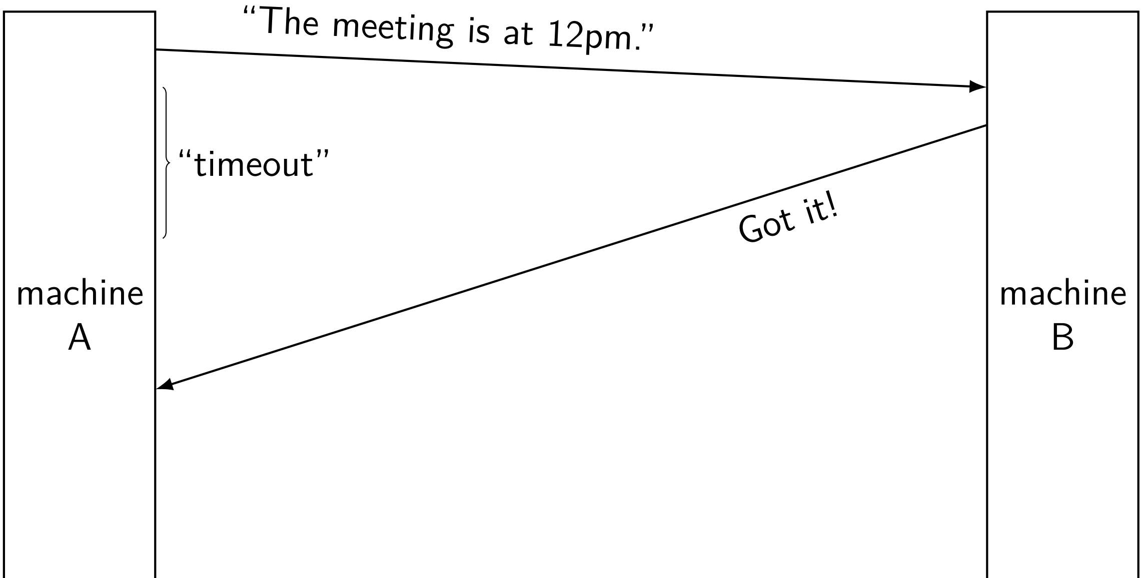


B resends, can’t tell message is just slow

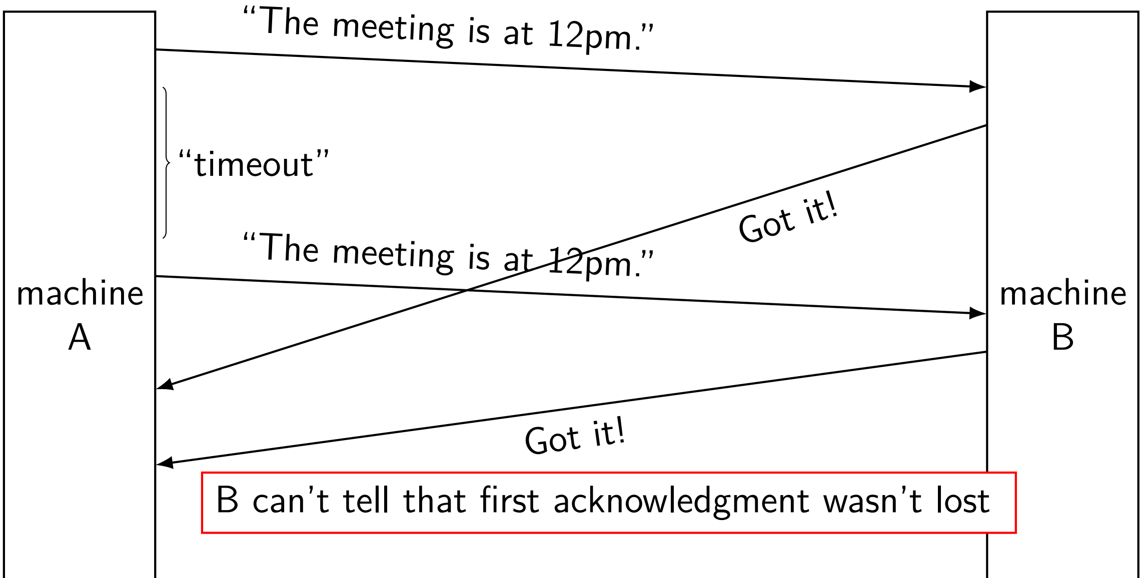
delayed acknowledgements



delayed acknowledgements



delayed acknowledgements



network limitations/failures

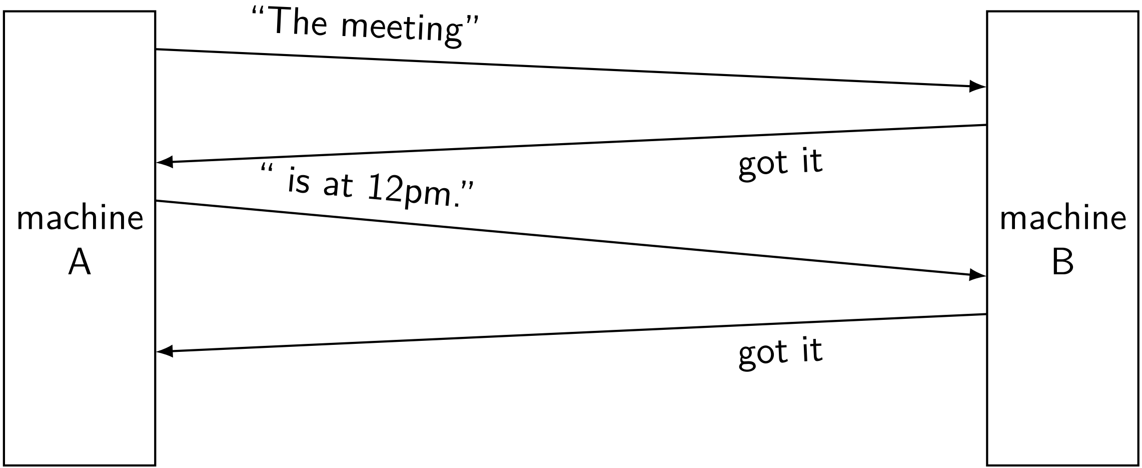
messages lost

messages delayed/reordered

messages limited in size

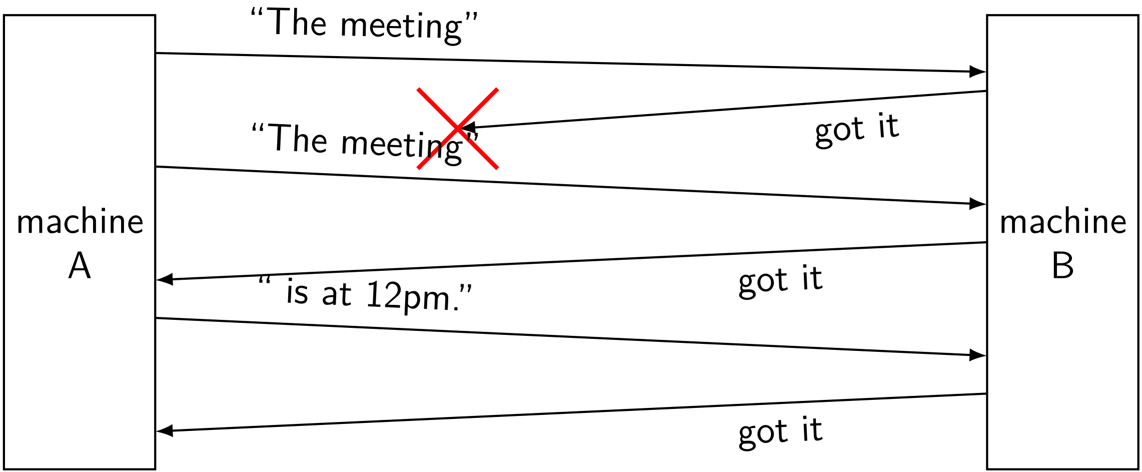
messages corrupted

splitting messages: try 1

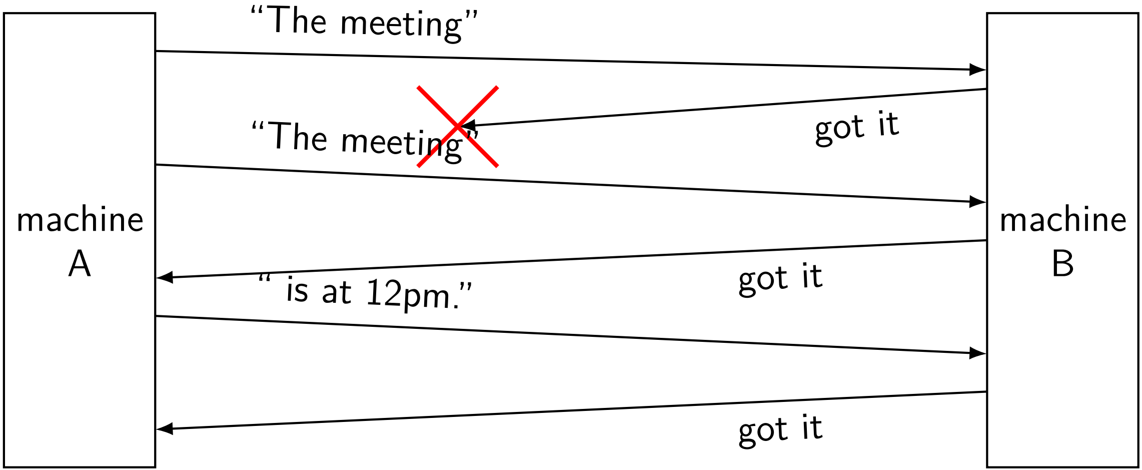


reconstructed message:
The meeting is at 12pm.

splitting messages: try 1 — problem 1



splitting messages: try 1 — problem 1



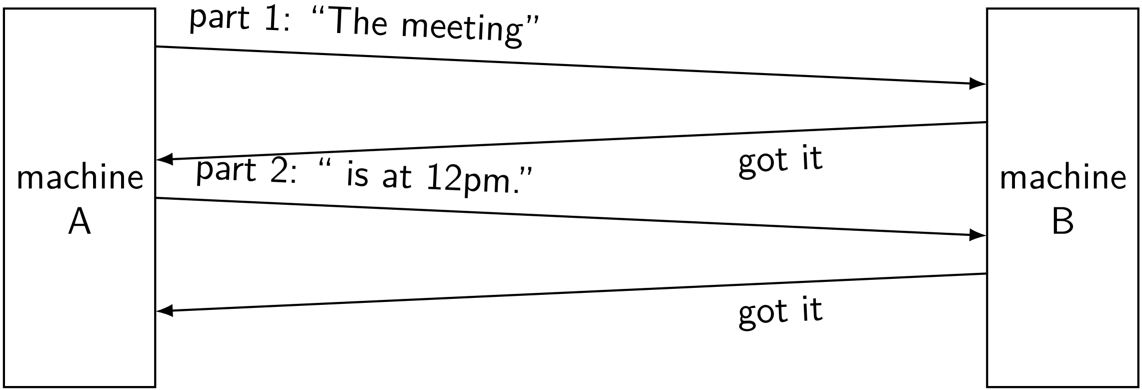
reconstructed message:
The meetingThe meeting is at 12pm.

exercise: other problems?

other scenarios where we'd also have problems?

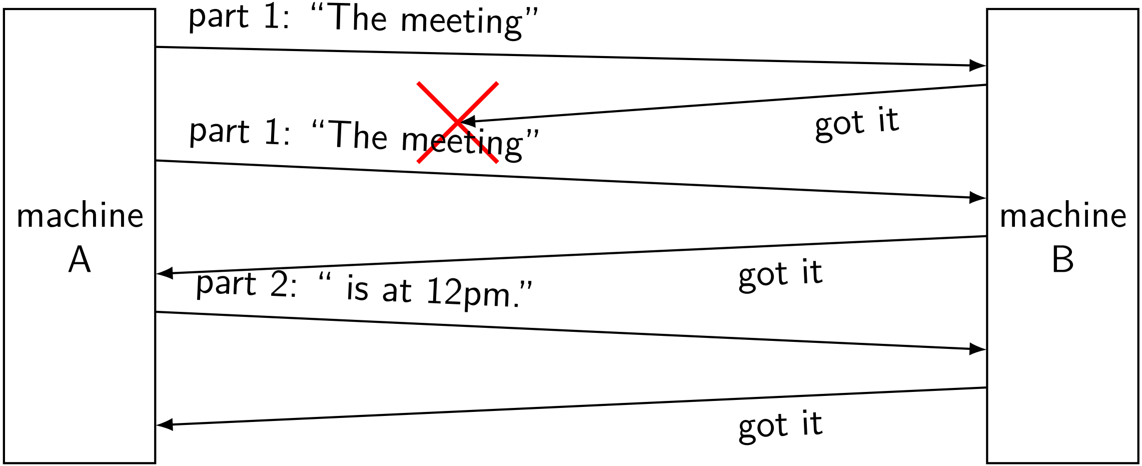
1. message (instead of acknowledgment) is lost
2. first message from machine A is delayed a long time by network
3. acknowledgment of second message lost instead of first

splitting messages: try 2



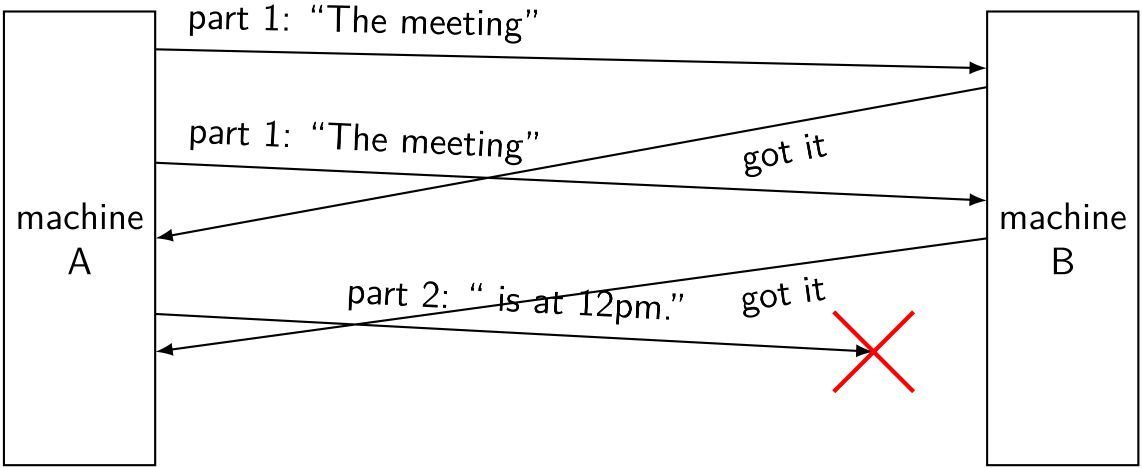
reconstructed message:
The meeting is at 12pm.

splitting messages: try 2 — missed ack



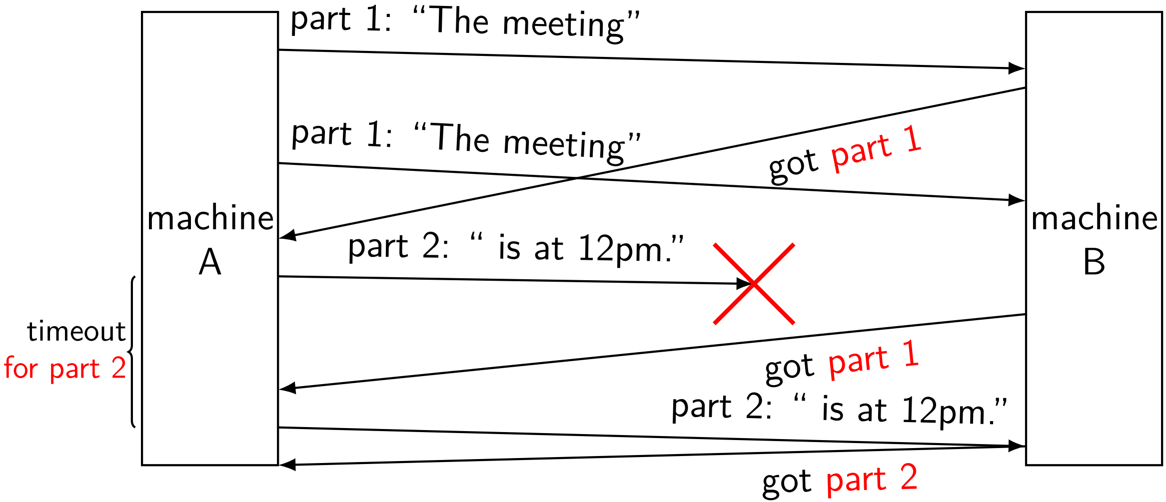
reconstructed message:
The meeting is at 12pm.

splitting messages: try 2 — problem



A thinks: part 1 + part 2 acknowleged!

splitting messages: version 3



network limitations/failures

messages lost

messages delayed/reordered

messages limited in size

messages corrupted

message corrupted

instead of sending "message"

say Hash("message") $=$ 0xABCDEF12

then send "0xABCDEF12,message"

when receiving, recompute hash

pretend message lost if does not match

“checksum”

these hashes commonly called “checksums”

in UDP/TCP, hash function: treat bytes of messages as array of integers; then add integers together

going faster

so far: send one message, get acknowledgments

pretty slow

instead, can send a bunch of parts and get them acknowledged together

need to do *congestion control* to avoid overloading network

backup slides