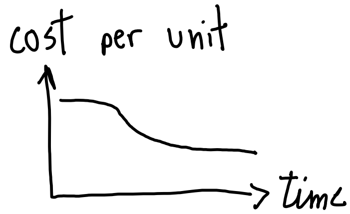
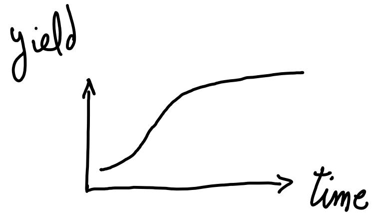
Cost



Manufacturing costs drop as expertise grows, for that process

-- better methods
-- better equipment
-- less waste (time, materials)



Yield = 1 - waste

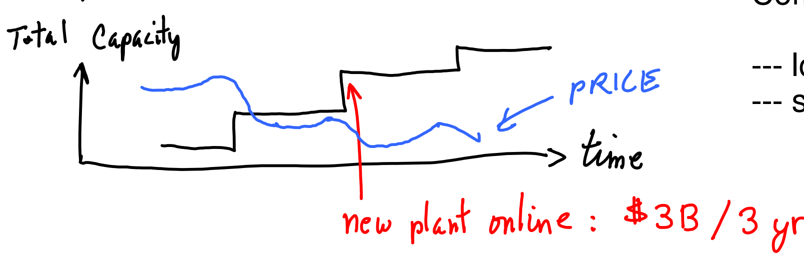
-- #(devices sellable) versus #(devices produced)

-- #(devices sellable) versus (cost to produce them)

E.g. DRAM ⇒ price $= \propto$ cost

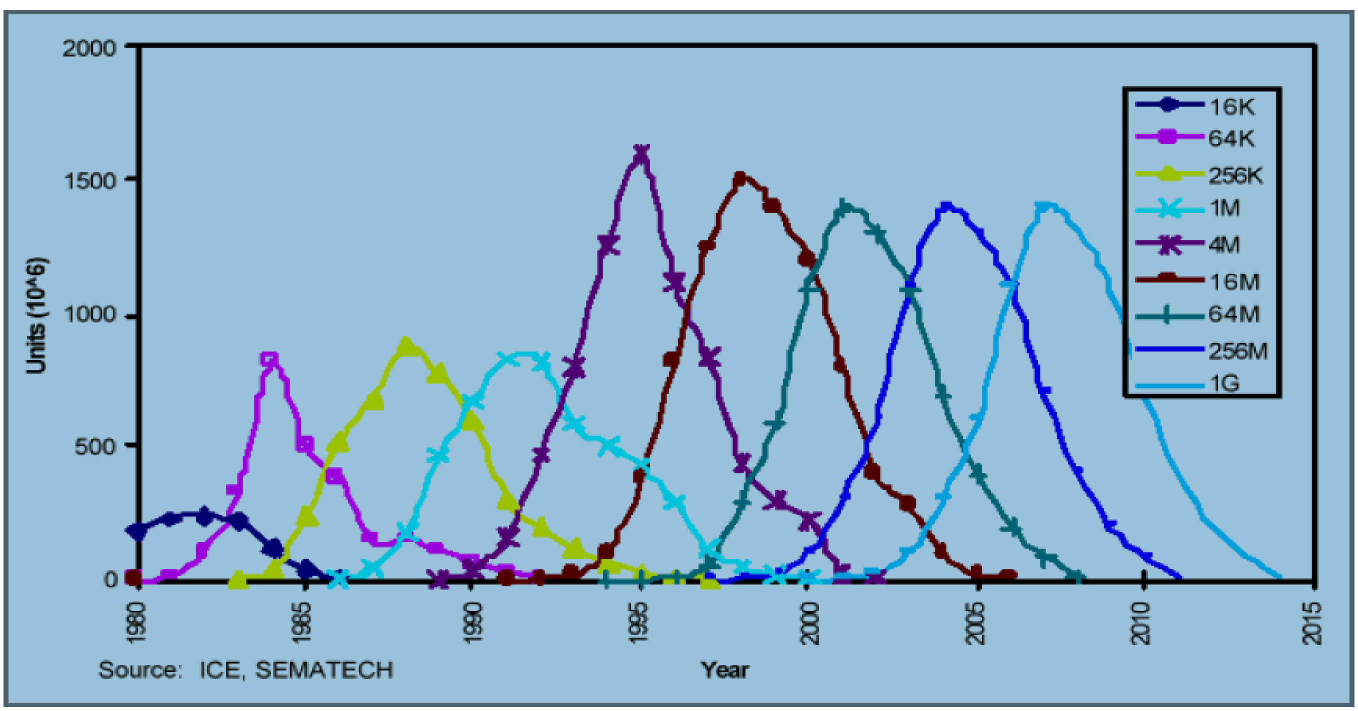
80% contract sales to large Equipment makers (hidden)

20% open market



Commodity market

--- lots of vendors
--- selling same items



DRAM Unit Volume by Generation [37]

Invest in largest demand ==> production cost amortized ==> larger profit
hot-new ==> high price / low volume ==> old-standard ==> low price

Costs Drop

$50/64

$65/256

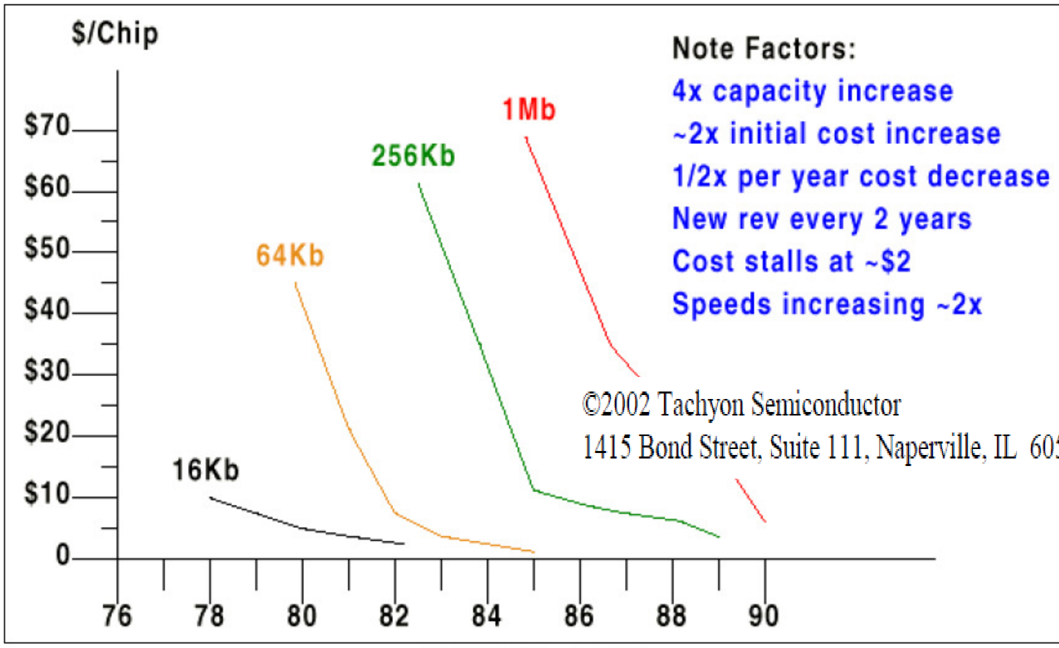
4x capacity

2x speed

Cost per bit

per bit/sec

Drop faster



DRAM Costs [45]

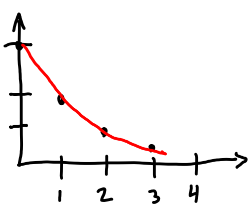
<u>Changes</u> Telecom (routers/switches) 20% ↑ ⇒ 50%

latency ↓ vs bandwidth ↑

(PCs)

SRAM	DRAM
12 ns	40 ns
$50/~2MB	$200/GB

Cost



Volume (log)

½x Cost : 2x Volume

Volume → supplier competition → lower Cost

⇒ Low-end Market (Price/Performance) ↓

Standardization / Volume ====> market acceptance of innovations

Cloud Pricing

AWS

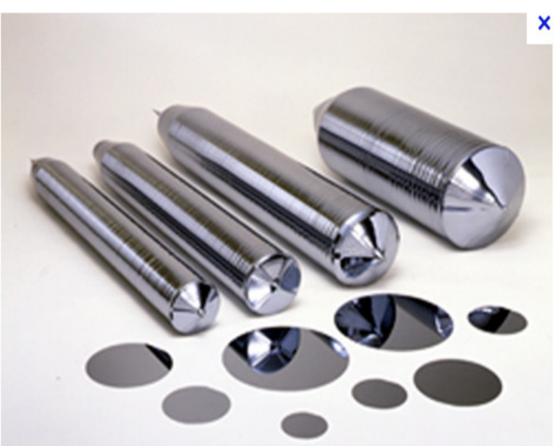
Combined efficiencies

Description	Type	CU	Original $ / CU / Hour	Current $ / CU / Hour	% Reduction	Aug 2006	Oct 2007	May 2008	Oct 2009	Feb 2010	July 2010	Sep 2010	Nov 2010	Nov 2011
Small - "the original"	m1.small	1	$0.10	$0.085	15%	$0.10			$0.09					
Large	m1.large	4	$0.10	$0.085	15%		$0.40		$0.34					
Extra Large	m1.xlarge	8	$0.10	$0.085	15%		$0.80		$0.68					
High-CPU Medium	c1.medium	5	$0.04	$0.03	15%			$0.20	$0.17					
High-CPU Extra Large	c1.xlarge	20	$0.04	$0.03	15%			$0.80	$0.68					
High-Memory Double Extra Large	m2.2xlarge	13	$0.09	0.077	17%				$1.20			$1.00		
High-Memory Quad Extra Large	m2.4xlarge	26	$0.09	0.077	17%				$2.40			$2.00		
High Memory Extra Large	m2.xlarge	6.5	$0.12	0.077	33%					$0.75				
Cluster Compute	cc1.4xlarge	33.5	$0.05	$0.04	19%						$1.60			
Cluster Compute Eight Extra Large	cc2.8xlarge	88	$0.03	$0.03	0%									$2.40
Micro	t1.micro	0.9	$0.02	$0.02	0%							$0.02		
Cluster GPU Instance	cg1.4xlarge	33.5	$0.06	$0.06	0%								$2.10	

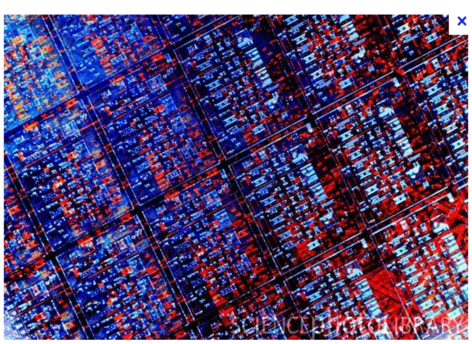
Price Reductions

CPUs, Chips, SOC

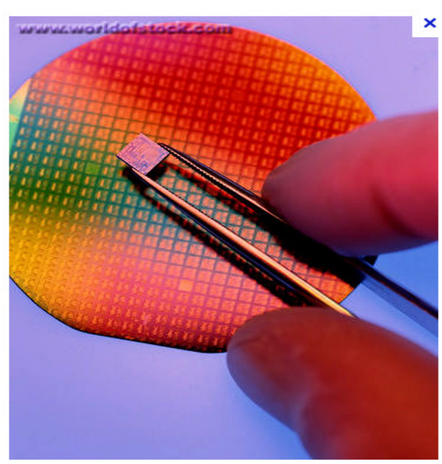
Si ingots slicing → wafers



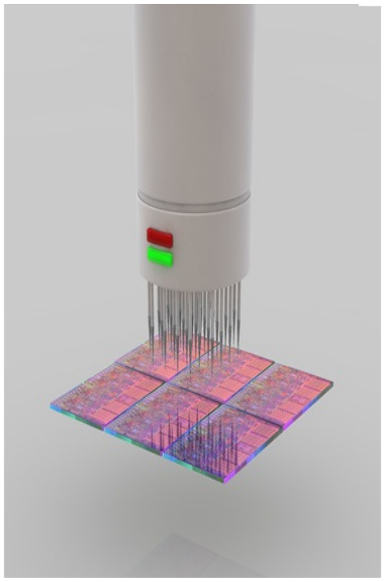
masking, etching, doping

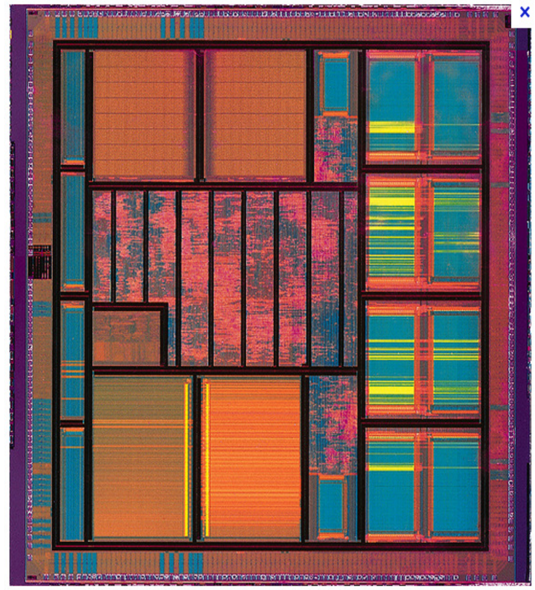


dicing →



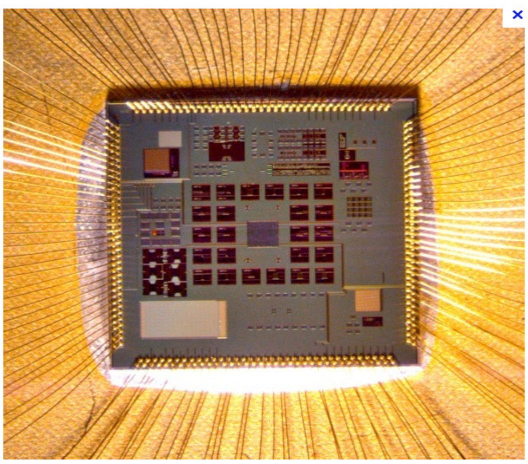
Circuit testing

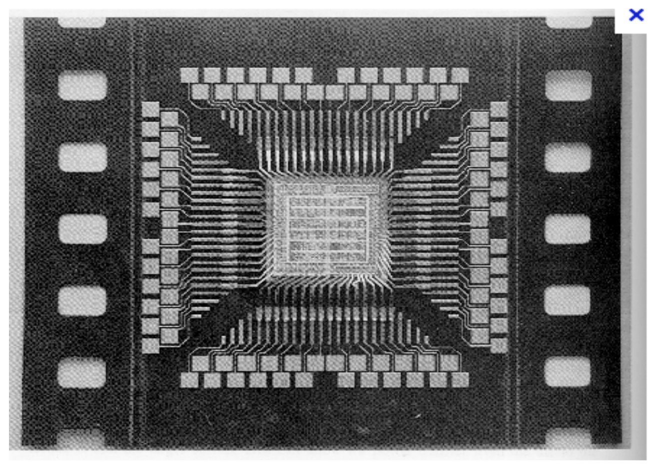




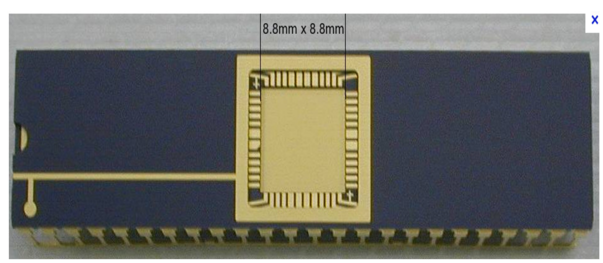
Pad Bonding

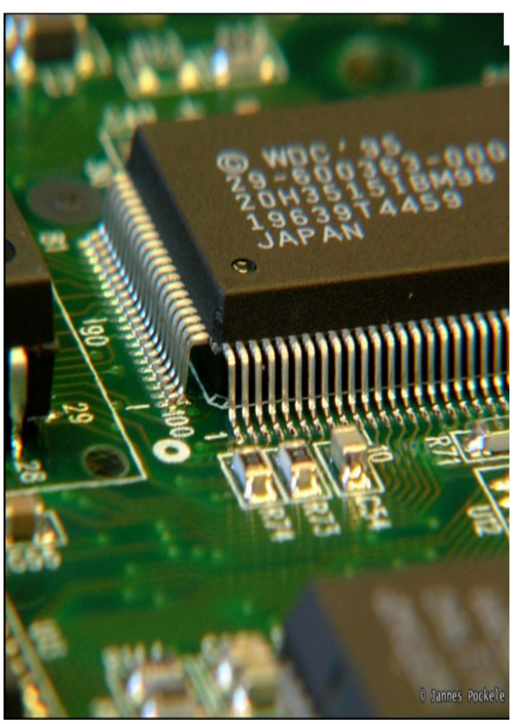
Pin Packaging





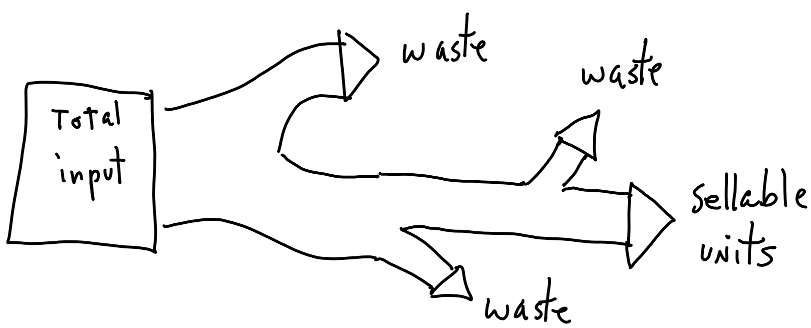
encasing



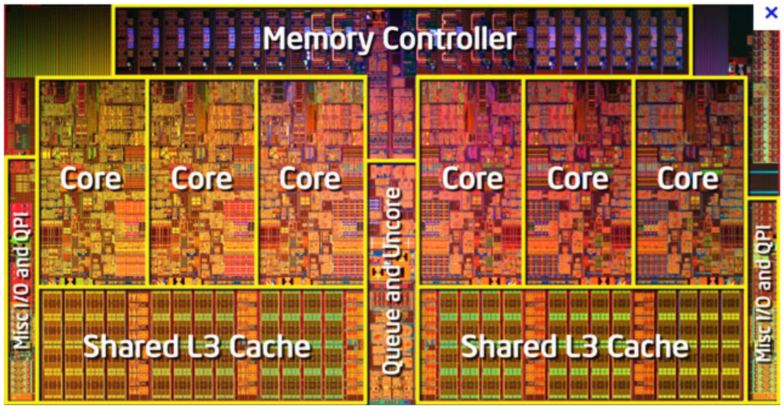


board printing

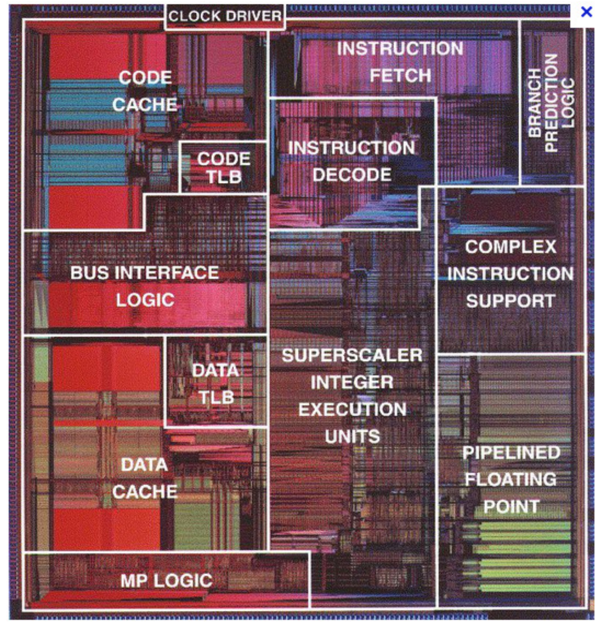
mounting



what's inside?



i7

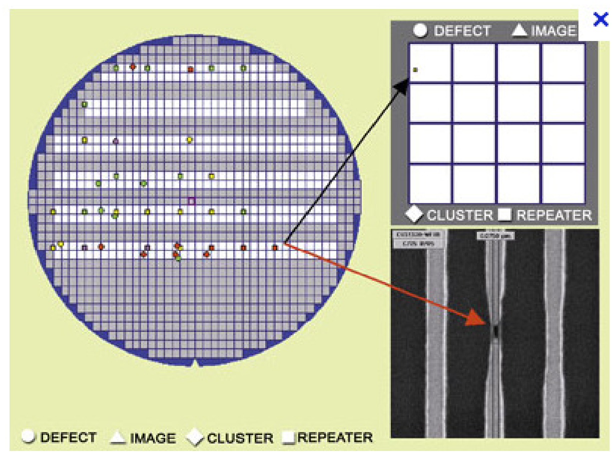


P5

$$\text{Cost} = \frac{C_{die} + C_{Test_1} + C_{package} + C_{Test_2}}{\#(\text{sellable units})}$$

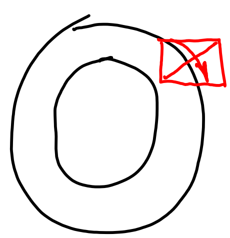
$$C_{die} = C_{wafer} / (\#\text{dies})(\text{yield})$$

$$C_{wafer} \approx \$5000$$



$$\#(\text{dies}) = \left(\frac{\text{Area wafer}}{\text{Area}_{die}}\right) - \left(\frac{\text{Circumference}_{Wafer}}{\text{Diagonal}_{die}}\right)$$

$$= \frac{\pi r^2}{A_{die}} - \frac{2\pi r}{\sqrt{2}\sqrt{A_{die}}}$$



$$\text{yield} \cong \frac{\#(\text{good wafers}) / \#(\text{wafers})}{\left[1 + \frac{\#(\text{defects})}{cm^2}(A_{die}\, cm^2)\right]^N}$$

Curve fitting $\Rightarrow$ $N \in [11.5, 15.5]$

$\frac{\#(\text{defects})}{cm^2} \approx 0.04$ ← function of time + volume

300 mm Wafer

$A_{die} = 2.25\ cm^2 \Rightarrow 109$ good

$A_{die} = 1\ cm^2 \Rightarrow 424$ good

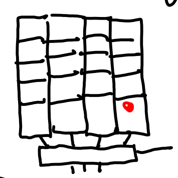
P5 Sandy Bridge	embedded CPU, 32b	printer Controller
20 x 10 mm	0.1 cm^2	0.04 cm^2
2 cm^2	\$13	\$0.1
\$50		

Amortized Costs

Mask = \$1M

⇒ reconfigurable gate arrays

Redundancy, e.g.



Select 3 banks

1 spare

$$\left.\begin{array}{l}\text{Die size} = \#(\text{Transistors}) + \#(\text{pins})\uparrow \\ + \\ \text{Volume}\downarrow \\ + \\ \text{Customization}\uparrow\end{array}\right\}\ \text{cost}$$

Warehouse-Scale Costs

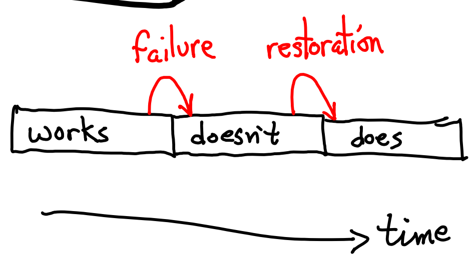
$$\text{Cost}_{\text{computing}} = \text{Cost}_{\text{equipment}}/\text{unit Time} + \text{Cost}_{\text{power}} + \text{Cost}_{\text{structure}}/\text{Time} + \text{Cost}_{\$}/\text{Time} + \text{Cost}_{\text{repair}}$$

$$\underbrace{\left(\frac{\text{Computers} + \text{networks}}{3\ \text{yr}}\right)}_{(60\%)} + \underbrace{\left(\quad \text{other} \quad\right)}_{(40\%)}$$

Failures, systemic, hierarchic

Service Accomplishment = it works

Service Interruption = it doesn't



Reliability = time to failure

Mean Time To Failure (MTTF) = Avg (Reliability) = Avg (does)

Mean Time To Repair (MTTR) = Avg (doesn't)

Failure Rate = $1/MTTF$

Mean Time Between Failures (MTBF) = MTTF + MTTR

Availability = % time working = $\dfrac{MTTF}{MTBF}$

Multi-level Recovery

- error correction
- redo
- redundancy

E.G., Disk sub-system

10 disks, MTTF = 1 Mhr

1 ATA controller, MTTF = $\frac{1}{2}$ Mhr

1 power supply, MTTF = $\frac{1}{5}$ Mhr

1 fan, MTTF = $1/5$ Mhr

1 ATA cable, MTTF = 1 Mhr

Assume: exponentially dist. failures

- independent of T
- No dependency between components

$E(\#\text{ Failures in } \Delta T) = \mu \, \Delta T$ ← avg failure rate

$1 \text{ failure} = \mu \, \overline{\Delta T_1}$

$MTTF = \overline{\Delta T_1} = 1/\mu$

$$\mu = \sum_i \mu_i = \sum_i (1/MTTF_i)$$

$$= (10 + 2 + 5 + 5 + 1)/Mhr = 23/Mhr$$

$$MTTF = \frac{10^6 \text{ hr}}{23} \approx \left(\frac{10^2}{25}\right) 10^4 = 40{,}000 \text{ hr}$$

$(365\,d)(24\,hr/d) \approx$

$(400)(100/4) = \frac{1}{4}(40{,}000)$

$\Rightarrow$ 4 yr.

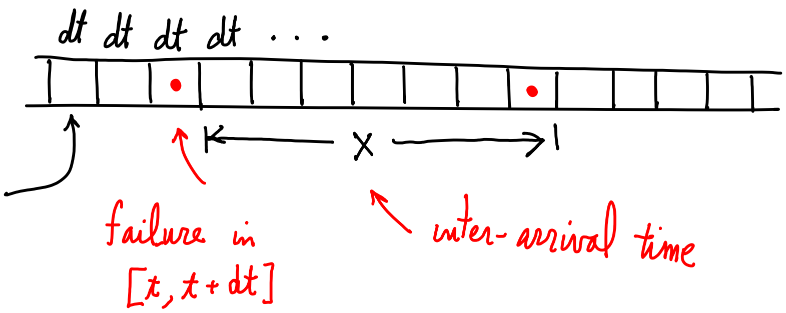
E.g., 2 power supplies?

Let's add a redundant power supply: if one fails we fix it while the other keeps working. How long before the system fails? That is, how long before we experience having the back-up fail before we can finish the repair?

Poisson

Bernoulli w/ $p = \lambda\, dt$

flip coin w/ Prob $= p = \lambda\, dt$



$$P(m, N, p) = \binom{N}{m} p^m q^{N-m}$$

$$\Rightarrow \frac{e^{-pN} (pN)^m}{m!}$$

let $pN = \lambda$

$$P(m, \lambda) = \frac{e^{-\lambda t} (\lambda t)^m}{m!}, \quad \lambda = \text{rate}$$

$$P(X_n = x) = 1 - e^{-\lambda x}$$

$$E(X) = 1/\lambda$$

1.1 Point Processes

Definition 1.1 *A simple point process $\psi = \{t_n : n \geq 1\}$ is a sequence of strictly increasing points*

$$0 < t_1 < t_2 < \cdots, \tag{1}$$

with $t_n \longrightarrow \infty$ as $n \longrightarrow \infty$. With $N(0) \stackrel{\text{def}}{=} 0$ we let $N(t)$ denote the number of points that fall in the interval $(0, t]$; $N(t) = \max\{n : t_n \leq t\}$. $\{N(t) : t \geq 0\}$ is called the counting process for ψ. If the t_n are random variables then ψ is called a random point process. We sometimes allow a point t_0 at the origin and define $t_0 \stackrel{\text{def}}{=} 0$. $X_n = t_n - t_{n-1}$, $n \geq 1$, is called the n^{th} interarrival time.

We view t as time and view t_n as the n^{th} *arrival* time. The word *simple* refers to the fact that we are not allowing more than one arrival to ocurr at the same time (as is stated precisely in (1)). In many applications there is a "system" to which customers are arriving over time

1.2 Renewal process

A random point process $\psi = \{t_n\}$ for which the interarrival times $\{X_n\}$ form an i.i.d. sequence is called a *renewal process*. t_n is then called the n^{th} *renewal epoch* and $F(x) = P(X \leq x)$ denotes the common interarrival time distribution. The *rate* of the renewal process is defined as $\lambda \stackrel{\text{def}}{=} 1/E(X)$.

- Poisson distribution can be derived by taking the appropriate limits of the binomial distribution

$$P(m, N, p) = \frac{N!}{m!(N-m)!} p^m q^{N-m}$$

$$\frac{N!}{(N-m)!} = \frac{N(N-1)\cdots(N-m+1)(N-m)!}{(N-m)!} = N^m$$

$$q^{N-m} = (1-p)^{N-m} = 1 - p(N-m) + \frac{p^2(N-m)(N-m-1)}{2!} + \cdots \approx 1 - pN + \frac{(pN)^2}{2!} + \cdots \approx e^{-pN}$$

$$P(m, N, p) = \frac{N^m}{m!} p^m e^{-pN} \qquad P(m, \mu) = \frac{e^{-\mu} \mu^m}{m!}$$

1.3 Poisson process

Definition 1.2 *A Poisson process at rate λ is a renewal point process in which the interarrival time distribution is exponential with rate λ: interarrival times $\{X_n : n \geq 1\}$ are i.i.d. with common distribution $F(x) = P(X \leq x) = 1 - e^{-\lambda x}$, $x \geq 0$; $E(X) = 1/\lambda$.*

$$P(N(t) = k) = e^{-\lambda t} \frac{(\lambda t)^k}{k!}, \quad k \geq 0.$$

k events in time t

Simulating a Poisson process at rate λ up to time T:

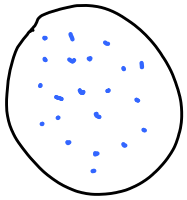
1. $t = 0$, $N = 0$
2. Generate U.
3. $t = t + [-(1/\lambda)\ln(U)]$. If $t > T$, then stop.
4. Set $N = N + 1$ and set $t_N = t$.
5. Go back to 2.

Suppose that for a Poisson process at rate λ, we condition on the event $\{N(t) = 1\}$, the event that exactly one arrival ocurred during $(0, t]$. We might conjecture that under such conditioning, t_1 should be uniformly distributed over $(0, t)$. To see that this is in fact so, choose $s \in (0, t)$. Then

$$\begin{aligned}
P(t_1 \leq s | N(t) = 1) &= \frac{P(t_1 \leq s, N(t) = 1)}{P(N(t) = 1)} \\
&= \frac{P(N(s) = 1, N(t) - N(s) = 0)}{P(N(t) = 1)} \\
&= \frac{e^{-\lambda s}\lambda s e^{-\lambda(t-s)}}{e^{-\lambda t}\lambda t} \\
&= \frac{s}{t}.
\end{aligned}$$

Independent, Random faults

Pool



vs

individual

biased coin w/ Prob(working) = p

Prob(failed) = $(1-p)$

coin tosses, independent, constant p

$\Rightarrow X = \{X_1, X_2, \dots\}$ $\quad X_i = \begin{cases} 0, & \text{failure} \\ 1, & \text{working} \end{cases}$

individual toss outcome

<u>n-trials</u> $v \in \{0,1\}^n$ $\quad \text{Prob}(X=v) = p^r (1-p)^{n-r}$ $\quad [\#\text{ones}(v) = r]$

System state $\Rightarrow \delta(X) \in \{0,1\}$ $\Rightarrow \begin{cases} \text{System is working}: \delta(X) = 1 \\ \text{System has failed}: \delta(X) = 0 \end{cases}$

Series System State

$$\text{working}(X) = \begin{cases} 0, & \text{some } X_i = 0 \\ 1, & \text{all } X_i = 1 \end{cases} \Rightarrow \begin{array}{l} \text{Prob}(\text{working}) = p^n \quad [\#\text{ones}(X) = n] \\ \text{Prob}(\text{failed}) = 1 - p^n \end{array}$$

Parallel System

$$\text{working} = \begin{cases} 0, & \text{all } X_i = 0 \\ 1, & \text{some } X_i = 1 \end{cases} \Rightarrow \begin{array}{l} \text{Prob}(\text{failed}) = (1-p)^n \quad [\#\text{zeroes}(X) = n] \\ \text{Prob}(\text{working}) = 1 - (1-p)^n \end{array}$$

Take n devices, run them for t hrs

<u>k working</u> $\quad X_i = \begin{cases} 1, & \text{device}_i \text{ still working} \\ 0, & \text{device}_i \text{ failed} \end{cases}$

$$k = \sum X_i \qquad E(k) = \sum^n E(X_i) = np \qquad E(X_i) = 1 \cdot \text{Prob}(X_i = 1) + 0 \cdot \text{Prob}(X_i = 0) = p$$

Assuming measured $k \approx E(k) \Rightarrow p = k/n$

Assuming t is a unit time interval, let $X = \{x_1, x_2, x_3 \ldots, x_n\}$

$$x_i = \begin{cases} 1, & \text{device working during period } i \\ 0, & \text{device failed in period } i \end{cases}$$

$$\text{Prob(failure at } k) = \text{Prob(working for } k \text{ intervals)} \times \text{Prob(fail in } (k+1) \text{ interval)}$$

$$= p^k (1-p) = p^k q$$

$$E(k) = \sum_{k=1}^{\infty} k p^k q = q \sum_0^{\infty} k p^k = qp \sum \frac{d}{dp} p^k = qp \frac{d}{dp} \sum^{\infty} p^k = qp \frac{d}{dp}\left(\frac{1}{1-p}\right)$$

$$= qp/(1-p)^2 \qquad\qquad \frac{d}{dp}(1-p)^{-1} = -(1-p)^{-2} \frac{d}{dp}(-p) = (1-p)$$

$$= qp/q^2 = p/q$$

$= MTTF$ e.g. $p = \frac{1}{2} \Rightarrow MTTF = (\frac{1}{2})/(\frac{1}{2}) = 1$

$q = \frac{1}{2^n} \Rightarrow MTTF = \frac{1-2^{-n}}{2^{-n}} = 2^n - 1$

2 power supplies

$$\text{Prob(either fails)} = (1-p^2) = Q$$

$$MTTF_2 = P/Q = \frac{1-(1-p^2)}{(1-p^2)} = \frac{p^2}{1-p^2} = \frac{p^2}{(1-p)(1+p)} = \frac{p}{q}\frac{p}{(1+p)}, \quad \text{for } p \approx 1$$

$$\approx \frac{p}{q}\left(\frac{1}{2}\right) = \frac{1}{2} MTTF_1$$

Prob(2^{nd} does fail in MTTR)

$$\approx \frac{MTTR}{MTTF_1}$$

$$P \approx p\,\Delta T \qquad p = \lambda\, dt$$

$$p = \frac{1}{MTTF} dt$$

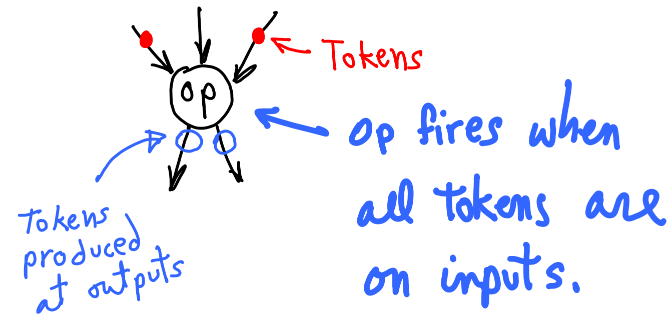
$$P \approx \frac{1}{MTTF_1}(MTTR)$$

$$P(x) = P_{1-2}(x - MTTR) P(MTTR)$$

$$E(x) = \frac{1-P}{P} = \frac{1 - \left(\frac{2}{MTTF}\right)\left(\frac{MTTR}{MTTF}\right)}{\left(\frac{2}{MTTF}\right)\left(\frac{MTTR}{MTTF}\right)} \approx \frac{(MTTF)^2}{2\, MTTR}$$

Tagged Dataflow Arch.

Dataflow Graph



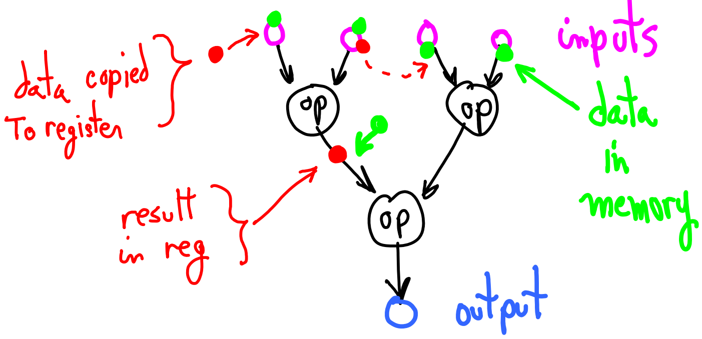
Data Dependency Graph

--- **Data** is copied from **memory** to **registers:**
red pebble on **green**

--- Operation done on register contents

--- **Result** written to **register:**
red pebble on arc

--- **Register reuse**: copy **registers** to **memory**:
green pebble on arc



Some portion of Data Graph pebbled red or green during computation.
Outputs pebbled green at completion

ID programming (LISP-like)

```
Def vsum A B =
    { C = array (1,n) :
        { for i 1 to n
              C_i <- A_i + B_i
        }
      In C
    }
```

Functional Programming

Any step can be taken whenever ready

C allocated in parallel with Loop unrolling

Function returns before C is filled

Each C_i waits for A_i and B_i

Semantics are write once

Synchronization is implicit

individual tasks distributed as HW available

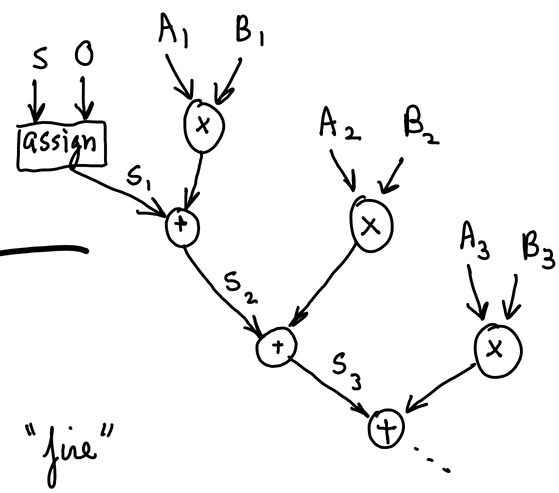
```
Def ip A B
  { s = 0
    In {
      For i 1 To n
        Next s = s + A_i x B_i

      finally s
    }
  }
```

Loop unrolling

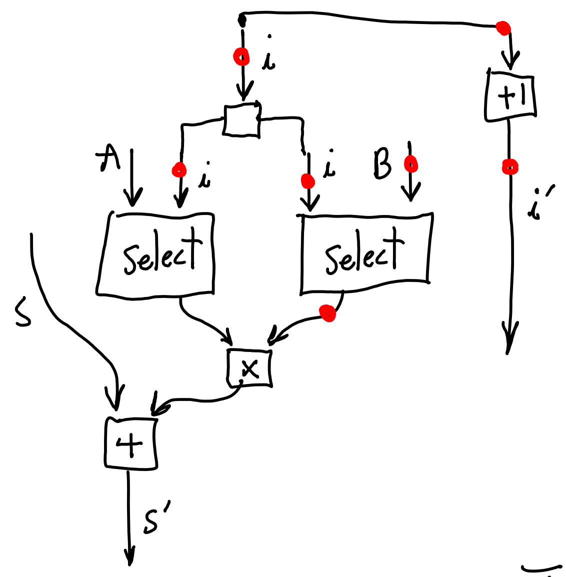
s_i+1 = s_i + A_i + B_i

Synchroniztion?

Write-Once semantics

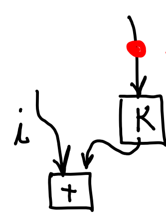


Dataflow Graph



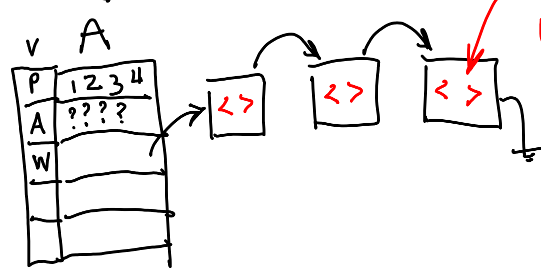
Tokens "fire" operators

Constants



Trigger token, no data

I-structure



READ Tokens, Waiting for data

Read token $\bullet \equiv$ $\langle addr_{data},$ READ, $addr_{inst} \rangle$

WRITE token $\bullet \equiv$ $\langle addr_{data},$ WRITE, value$\rangle$ → write data:

if v = Present → error

if v = Waiting, wakeup list

set v = Present

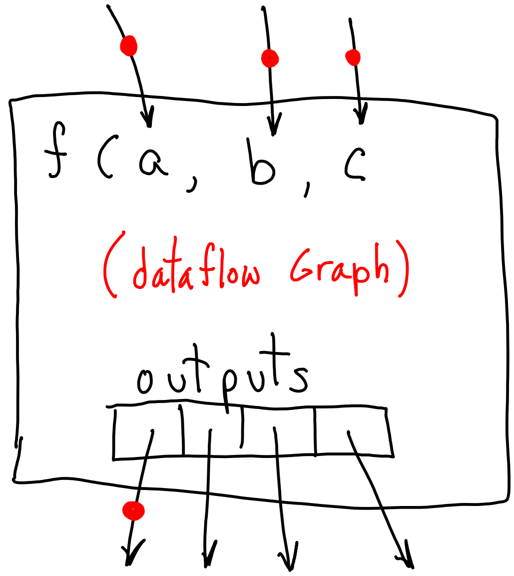
Split-phase operation: asynchronous R/W requests sent, asynchronous result returns value.

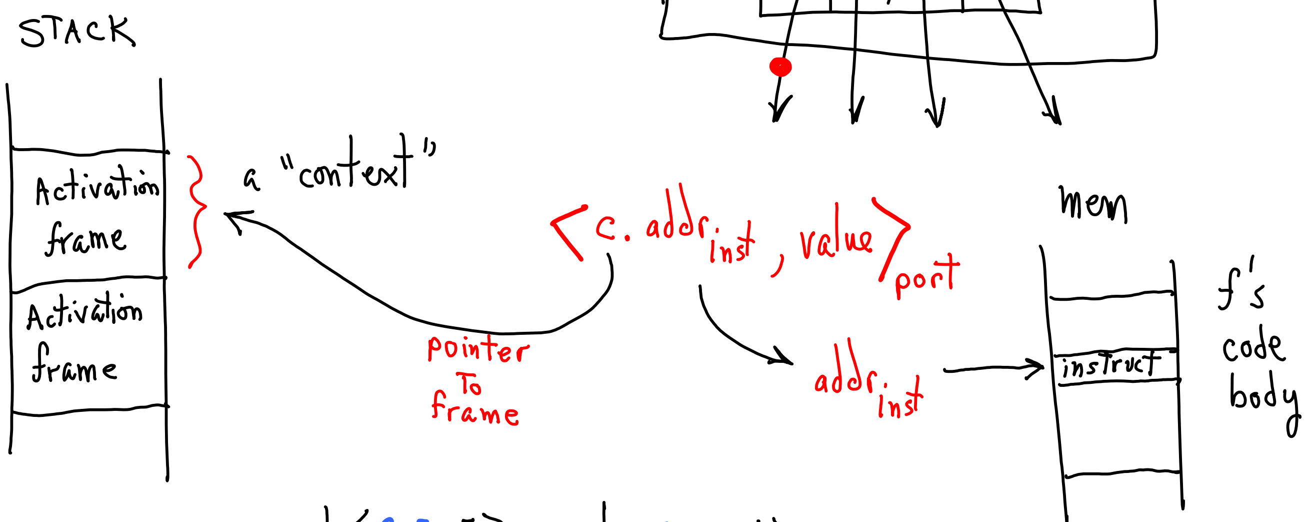
Good at hiding memory latency?

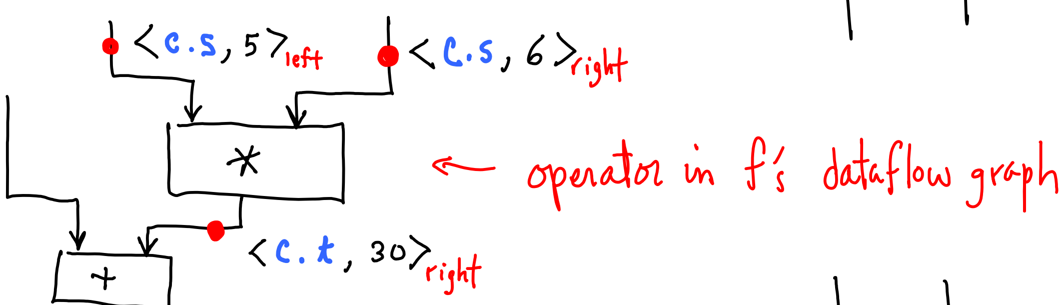
Good at tolerating high synchronization costs?

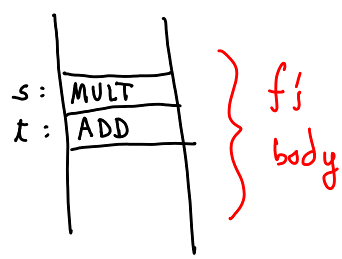
Functions / context

recursive calls	multiples
multiple parallel active calls	f applied in parallel, e.g. loop unrolling









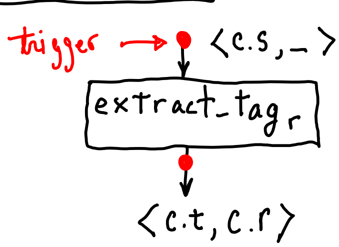
Firing Rule

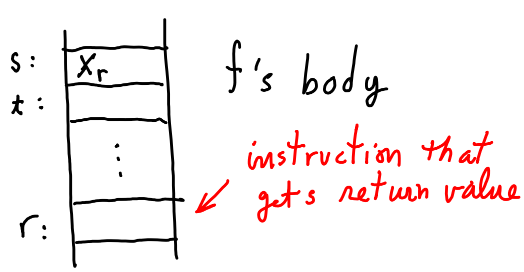
Operator fires when tags on inputs match

⇒ multiple tokens w/ different tags may be present

Calls

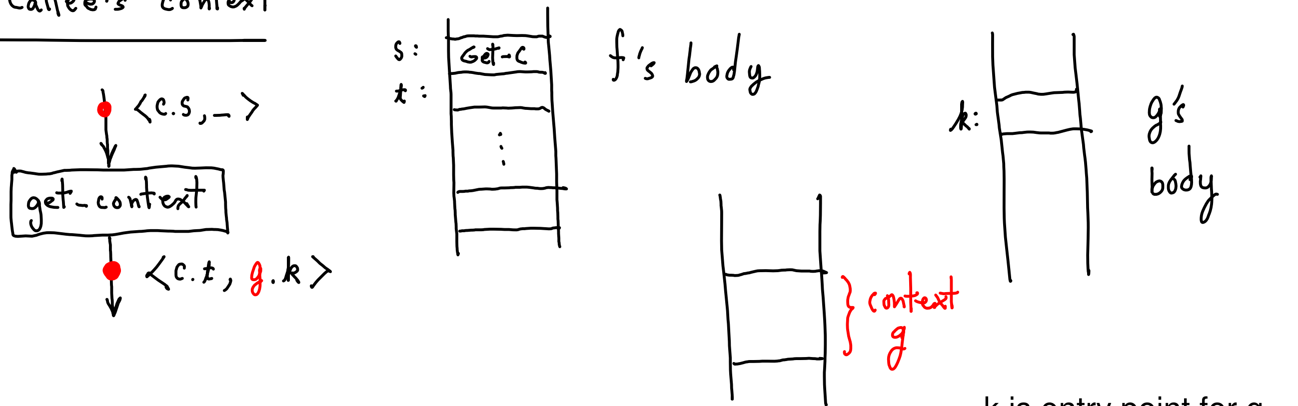
1. f sets return





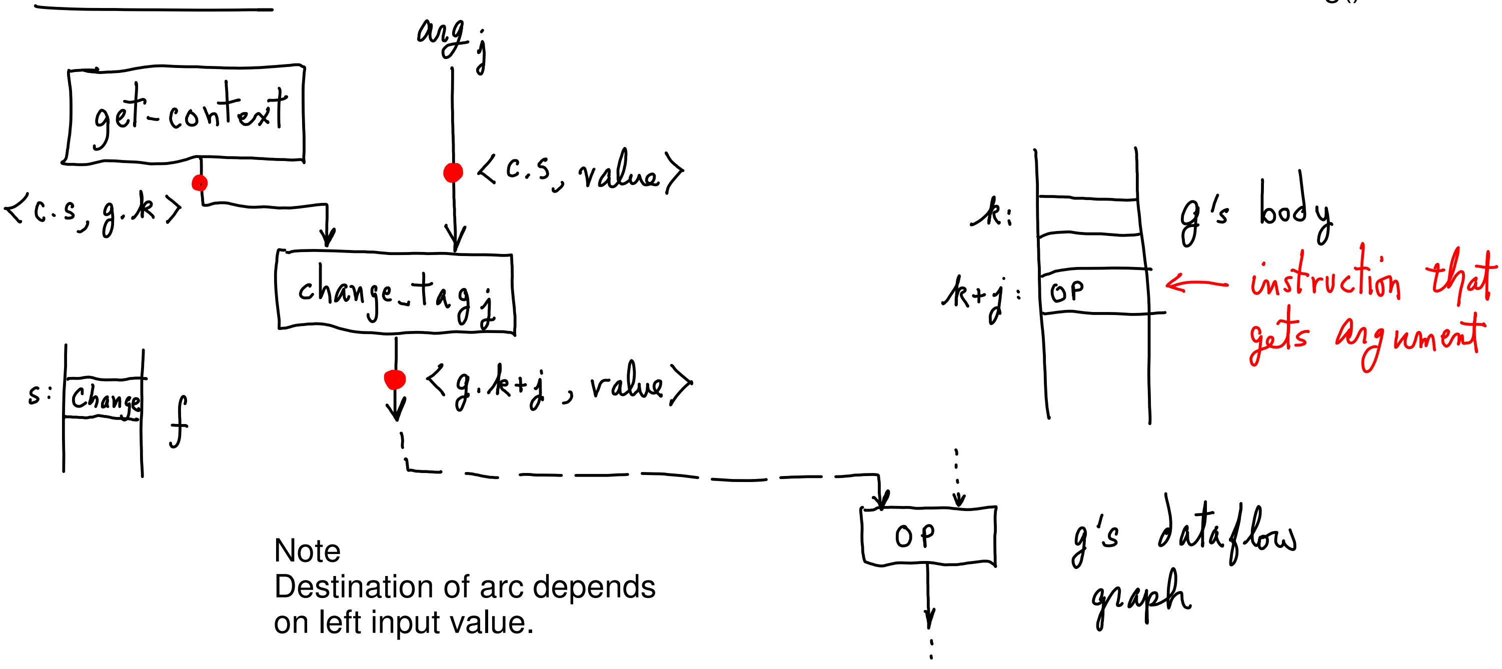
Note
"s" is current instruction,
"t" is next instruction

2. Get Callee's context



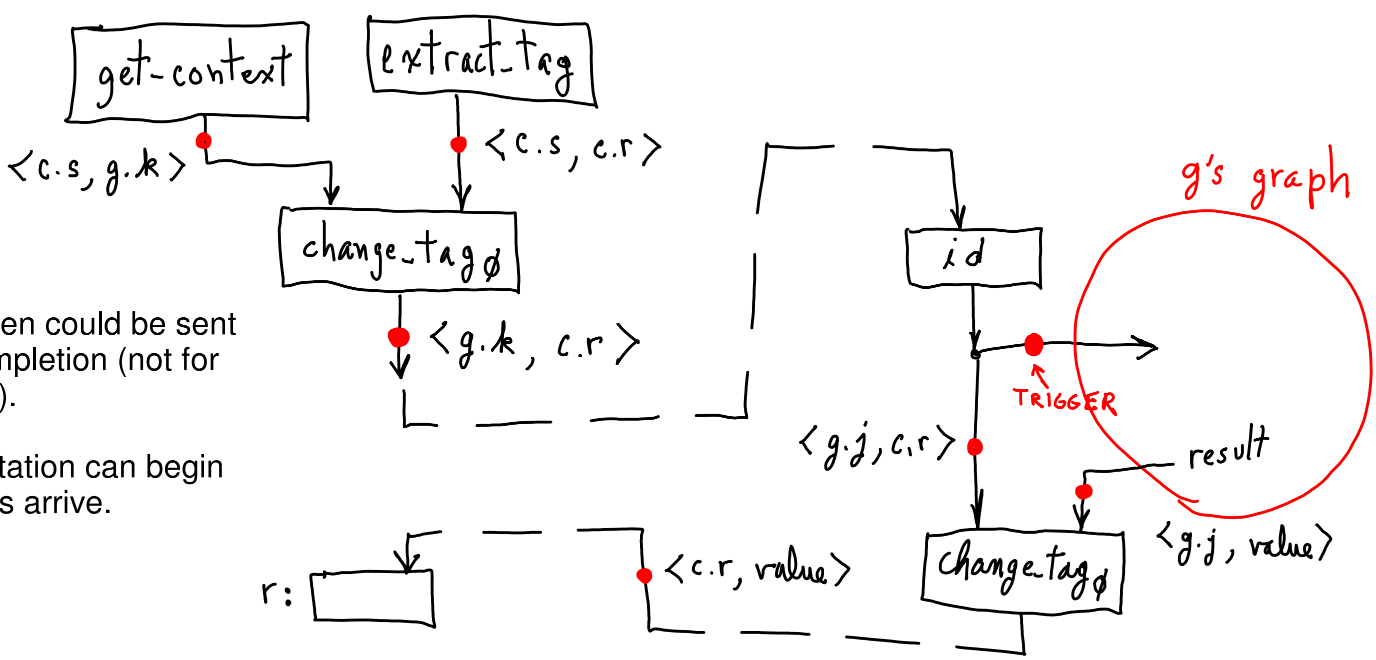
k is entry point for g.
context "g" is for this
invokation of g().

3. Pass args



Note
Destination of arc depends
on left input value.

3. Invoke function

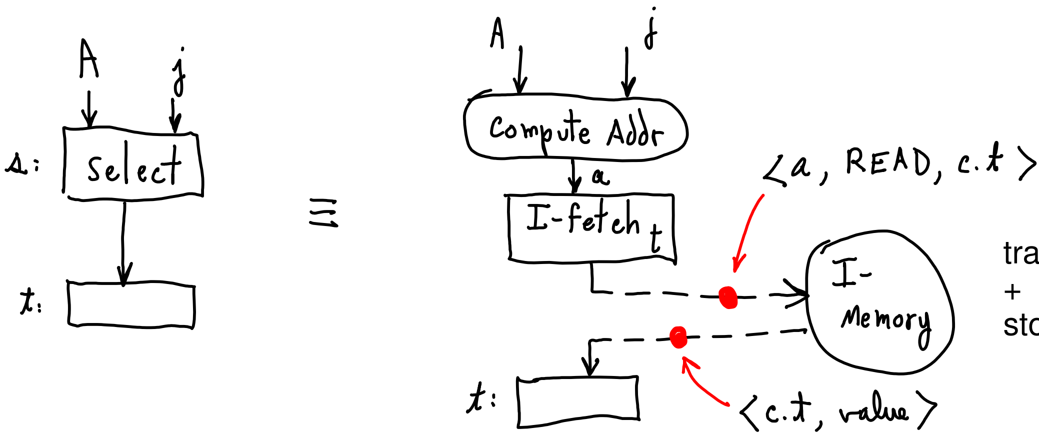


Note:
Return token could be sent
before completion (not for
g() though).

g's computation can begin
before args arrive.

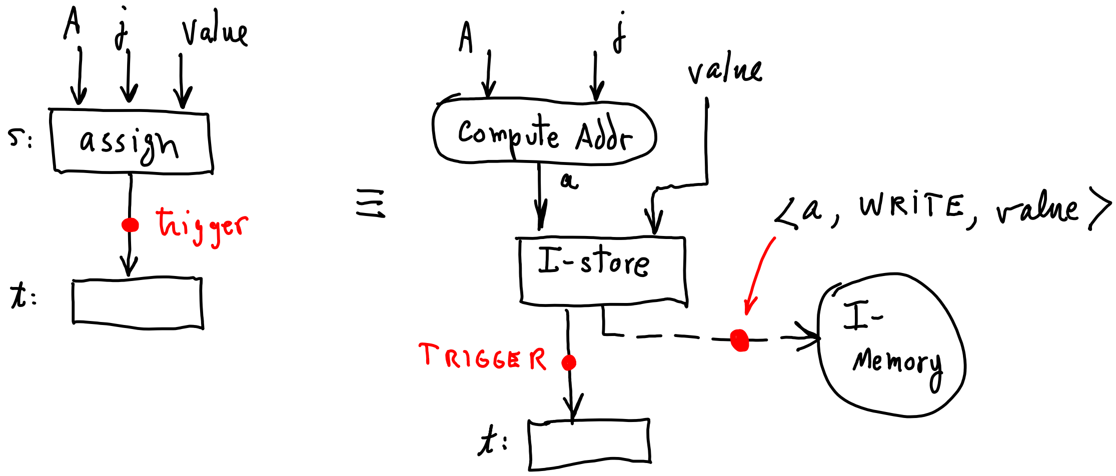
I-Structure R/W/allocate

Read

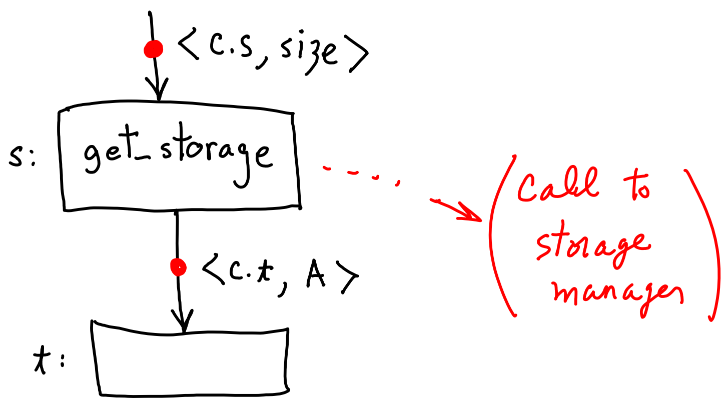


transaction processor
+
storage

Write



allocate



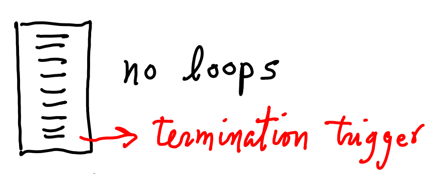
Can have other R/W protocols (not Write-Once only)

Can have active memory: operations in memory unit.

De-allocate

Cannot de-allocate while tokens exist in function call for context i.
==> Well-behaved graphs
1. initially, no tokens
2. given one token on every input, one token produced per output
3. after all output tokens produced, graph is empty.

Allocate by code blocks



by function body (same)

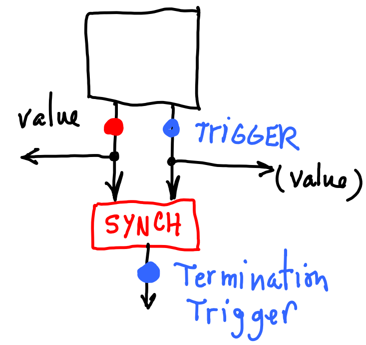
Call = allocate ⇒ code to processor
registers
context

Send trigger to callee (+ args)

Callee sends termination Trigger (+ return value)

Deallocate

conditional output



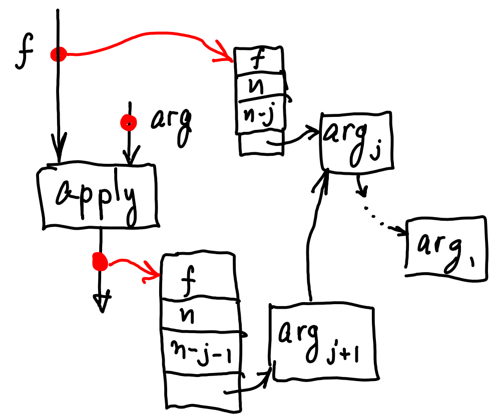
Functional APPLY

Def vsum A B
{
...
ln C }

⇒ move_point = vsum A

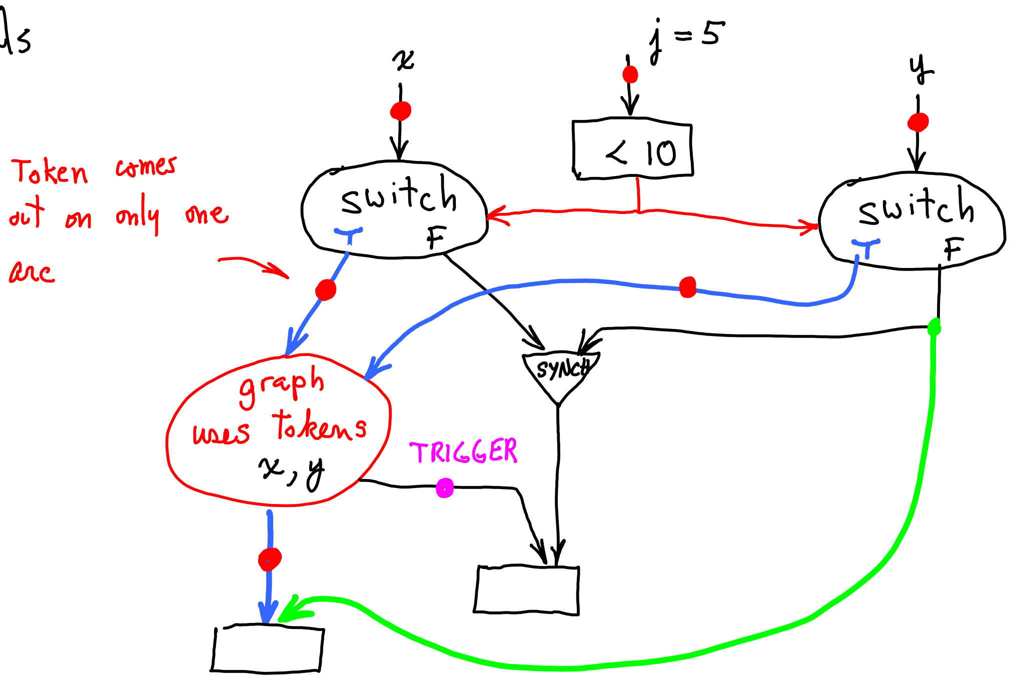
move_point B

evaluation occurs when all args present



Conditionals

if (j < 10)
x + y
else
y



Well behaved

Trigger output tokens have same address

Value output tokens have same address

Triggers only when all inputs have arrived and output token is ready