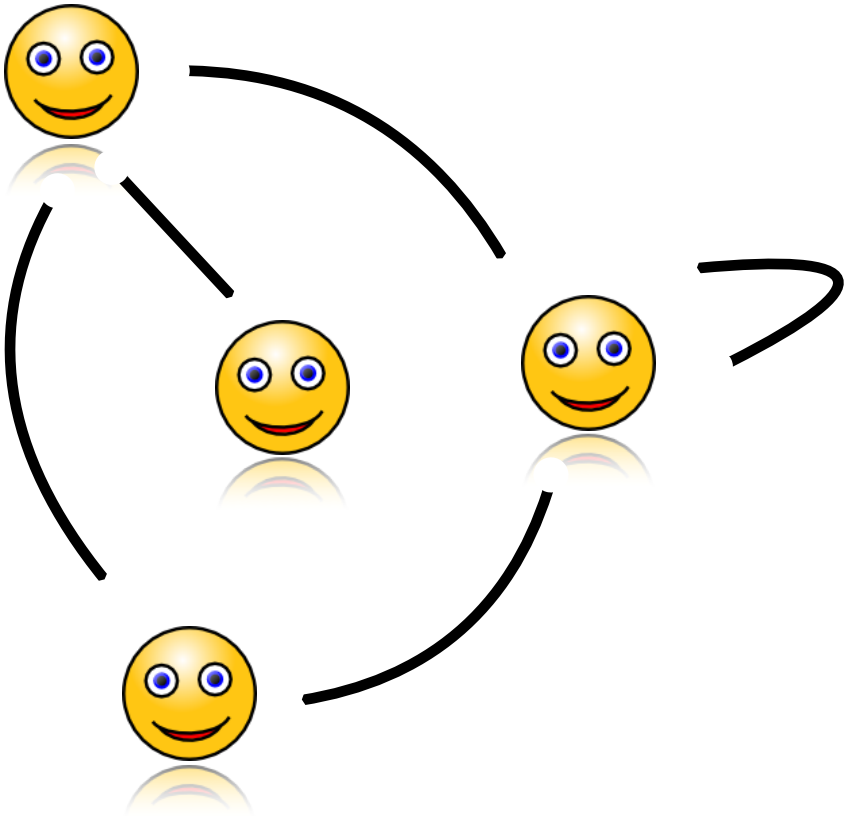
A distributed abstract machine

Computing with Collaborative Side Effects

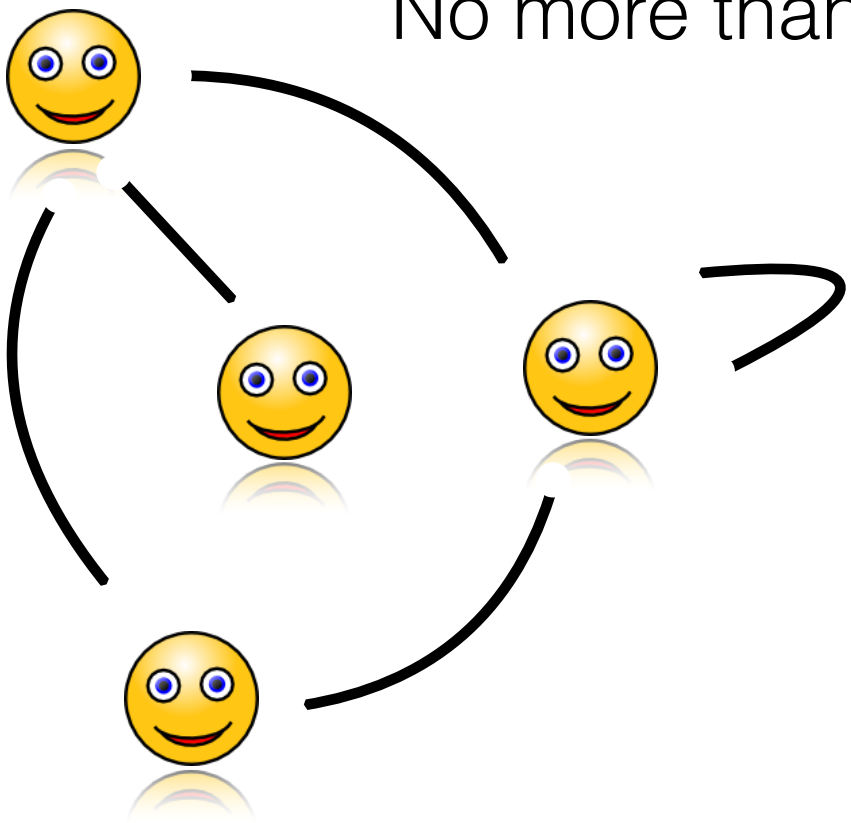
The machine

Graph



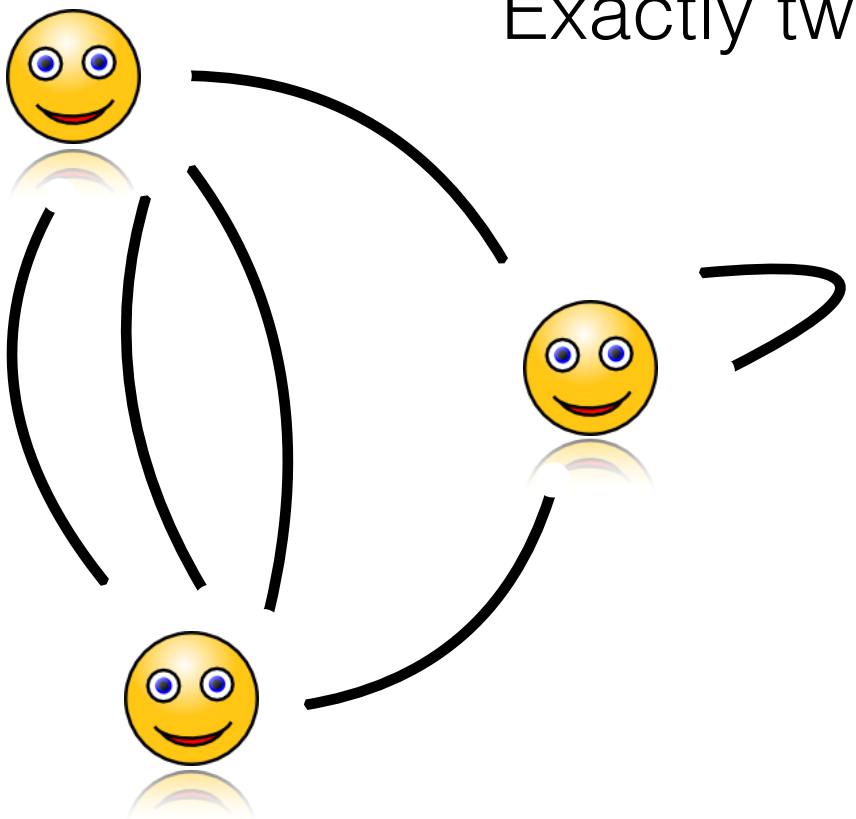
Limit of successors

No more than two successors

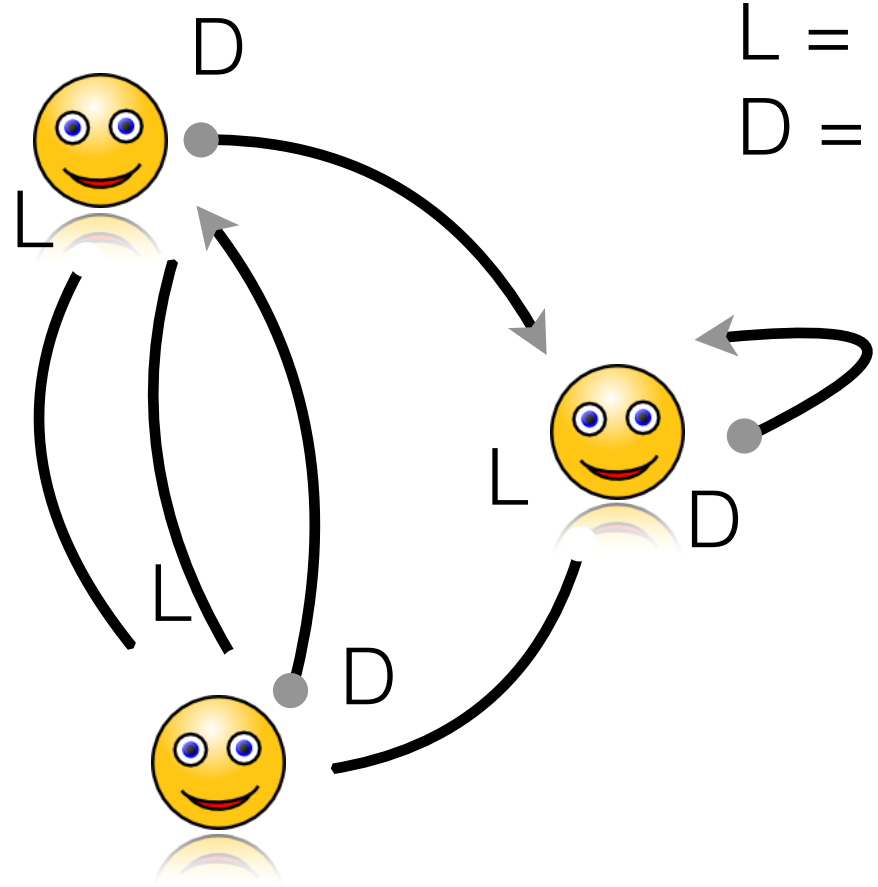


binary graph

Exactly two successors



Naming the two successors

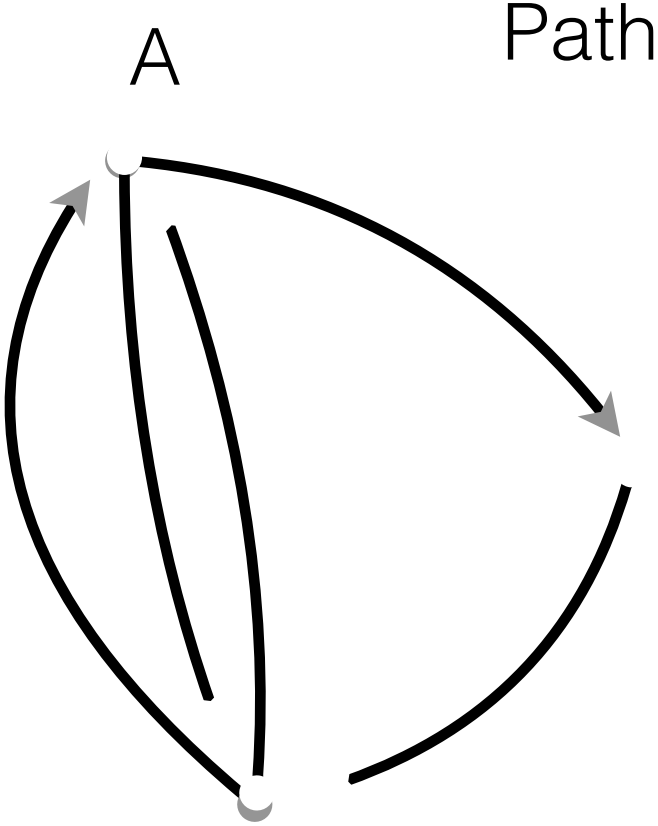


L = Light/Left
D = Dark/Derecho/cDr

A binary tree without leaves

Node ≜ Node x Node

Paths



Path : follow one or more edges

$\Delta = (L \mid D)^* = \delta^*$

Node x Path → Node

A x DDDLL → A

A x LD → A

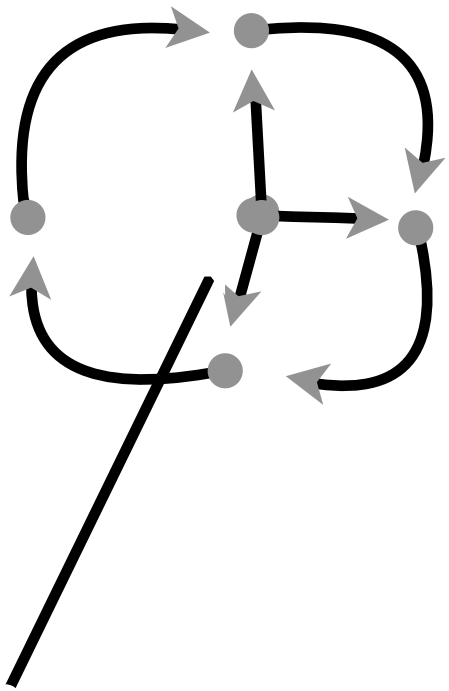
Instructions

If x is the central node, we want to do x.D = x.D.D

If x execute this instruction : this.D = this.D.D

We simplify with D👉DD

Instr ≜ Δ👉Δ



LD👉LDD

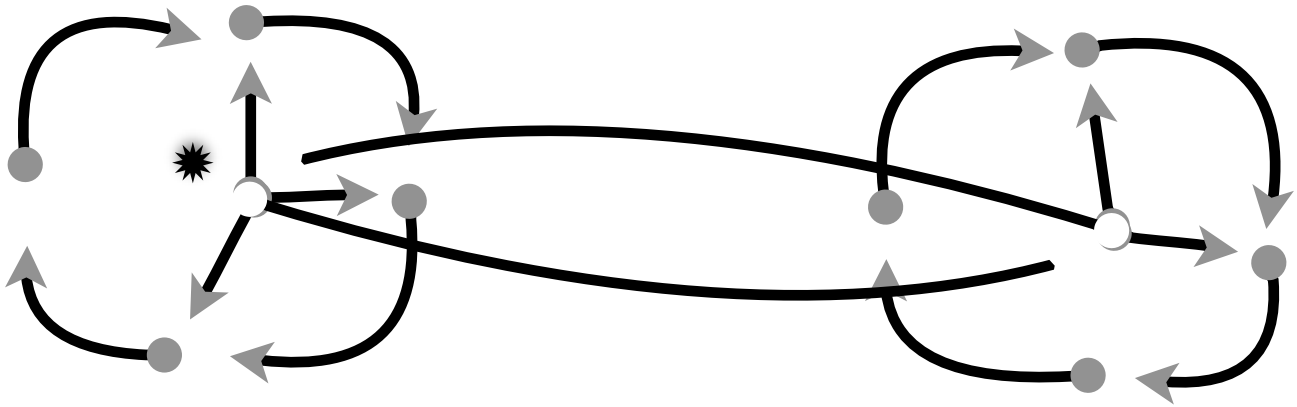
Only one fixed instruction per node

When instructions are applied ?

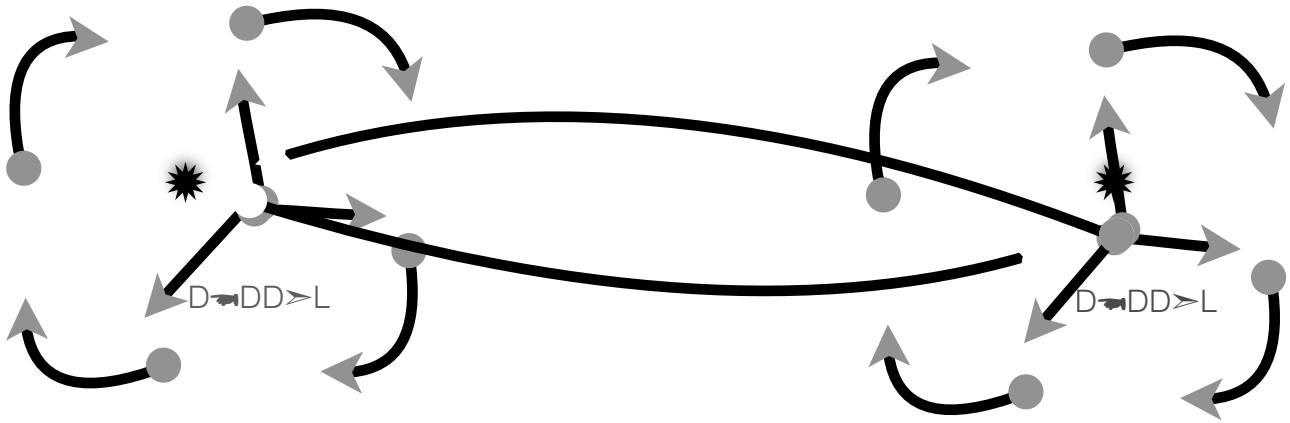
Tokens

Instruction is executed when a node receive a token. The token is given to an another node after execution

Instr ≜ D☚DD➢L



More tokens



Property: number of nodes, edges and tokens are preserved by execution

The machine

$\iota \in \text{Label}$ *a memory adress, an IP adress, …*

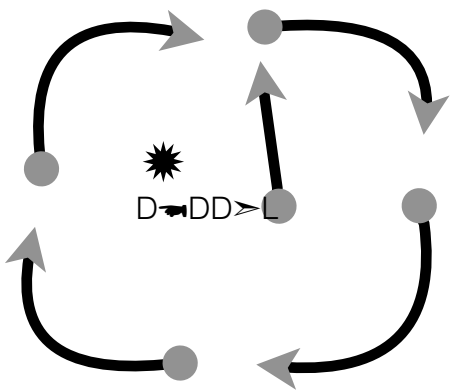
$\Delta \in \text{Path} \triangleq (L \mid D)^*$

$c \in \text{Instr} \triangleq \text{Path} \multimap \text{Path} \succ \text{Path}$

$\text{Node} \triangleq \text{Nat} \times \text{Label} \times \text{Label} \times \text{Instr}$

$\mu \in \text{Mach} \triangleq \text{Label} \rightarrow \text{Node}$

μ



$\mu(\iota_0) = \langle 1, \iota_0, \iota_1, D \multimap DD \succ L \rangle$

$\mu(\iota_1) = \langle 0, \iota_1, \iota_2, L \multimap L \succ L \rangle$

$\mu(\iota_2) = \langle 0, \iota_2, \iota_3, L \multimap L \succ L \rangle$

$\mu(\iota_3) = \langle 0, \iota_3, \iota_4, L \multimap L \succ L \rangle$

$\mu(\iota_4) = \langle 0, \iota_4, \iota_1, L \multimap L \succ L \rangle$

Sequential semantics

Following a path

$\mathcal{F} : \text{Mach} \times \text{Label} \times \text{Path} \to \text{Label}$

$$\mathcal{F}(\mu,\iota,[]) = \iota$$

$$\frac{\mu(\iota)=\langle n,\iota_L,\iota_D,c\rangle}{\mathcal{F}(\mu,\iota,L\cdot\Delta) \to \mathcal{F}(\mu,\iota_L,\Delta)}$$

$$\frac{\mu(\iota)=\langle n,\iota_L,\iota_D,c\rangle}{\mathcal{F}(\mu,\iota,D\cdot\Delta) \to \mathcal{F}(\mu,\iota_D,\Delta)}$$

Node access

$$f[x \leftarrow v] \triangleq \lambda y.\text{if } y=x \text{ then } v \text{ else } (f\ y)$$

$$\frac{\mu(\iota)=\langle n+1,\iota',\iota'',c\rangle}{(\text{decreaseToken}\ \mu\ \iota) = \mu[\iota \leftarrow \langle n,\iota',\iota'',c\rangle]}$$

$$\frac{\mu(\iota)=\langle n,\iota',\iota'',c\rangle}{(\text{increaseToken}\ \mu\ \iota) = \mu[\iota \leftarrow \langle n+1,\iota',\iota'',c\rangle]}$$

$$\frac{\mu(\iota)=\langle n,\iota',\iota'',c\rangle}{(\text{changeField}\ \mu\ \iota\ L\ v) = \mu[\iota \leftarrow \langle n,v,\iota'',c\rangle]}$$

rules for machine transition

$$\text{Mach} \longrightarrow \text{Mach}$$

$$\mu(\iota)=\langle n+1,\iota_L,\iota_D,\Delta_d \cdot \delta \twoheadleftarrow \Delta_s \twoheadrightarrow \Delta_n \rangle$$

$$\mu' = (\text{decreaseToken } \mu\ \iota)$$

$$\mu'' = (\text{changeField } \mu'\ \mathbb{F}(\mu,\iota,\Delta_d)\ \delta\ \mathbb{F}(\mu,\iota,\Delta_s))$$

$$\mu''' = (\text{increaseToken } \mu''\ \mathbb{F}(\mu,\iota,\Delta_n))$$

$$\mu \longrightarrow \mu'''$$

Turing completeness

Compiling SECD to the Distributed Abstract Machine

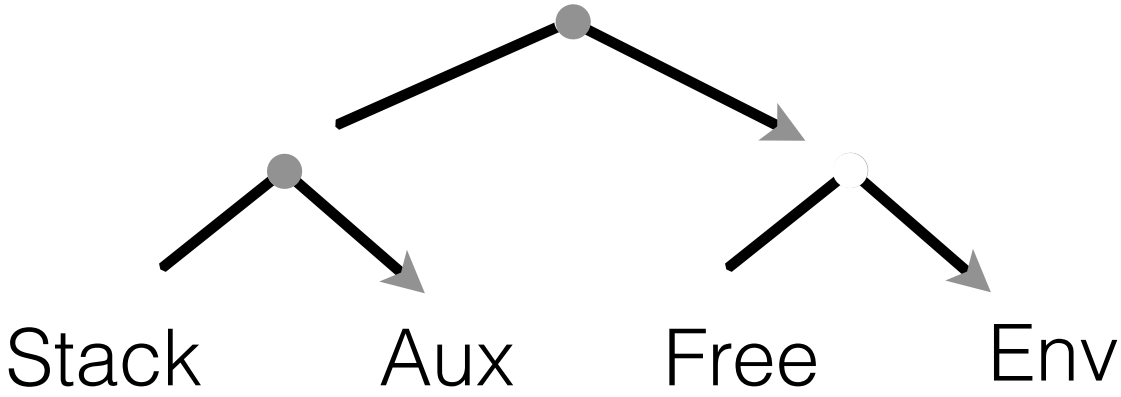
S(tack) *A stack of values*
E(nv) *A list of values*
C(ode) *A list of instructions*
D(ump) *A list of frames*

S(tack)
A(ux) *A register*
F(ree) *A free list*
E(nv)

No Dump

The code is outside

SAFE structure



S ≜ LL

A ≜ LD

F ≜ DL

E ≜ DD

18

Code structure

$\Delta_d \rightarrow \Delta_s \succ \Delta_n$ — *The SAFE structure*

Almost all the time Δ_n=D

S ≜ L**S**

A ≜ L**A**

F ≜ L**F**

E ≜ L**E**

SECD code to DAM

$$[\![\text{Code}]\!]_\iota \rightarrow \iota'$$

ι is the place where is the SAFE machine
ι' is the result in a form of a list linked by their dark field

$$[\![\text{nil}]\!]_\iota \rightarrow \text{<some stop>}$$

$$[\![i \,.\, c]\!]_\iota \rightarrow [\![i]\!]_\iota \bullet [\![c]\!]_\iota$$

Allocation

$\text{Alloc}(\iota, \Delta_L, \Delta_D, \Delta_r) \rightarrow \iota_S$

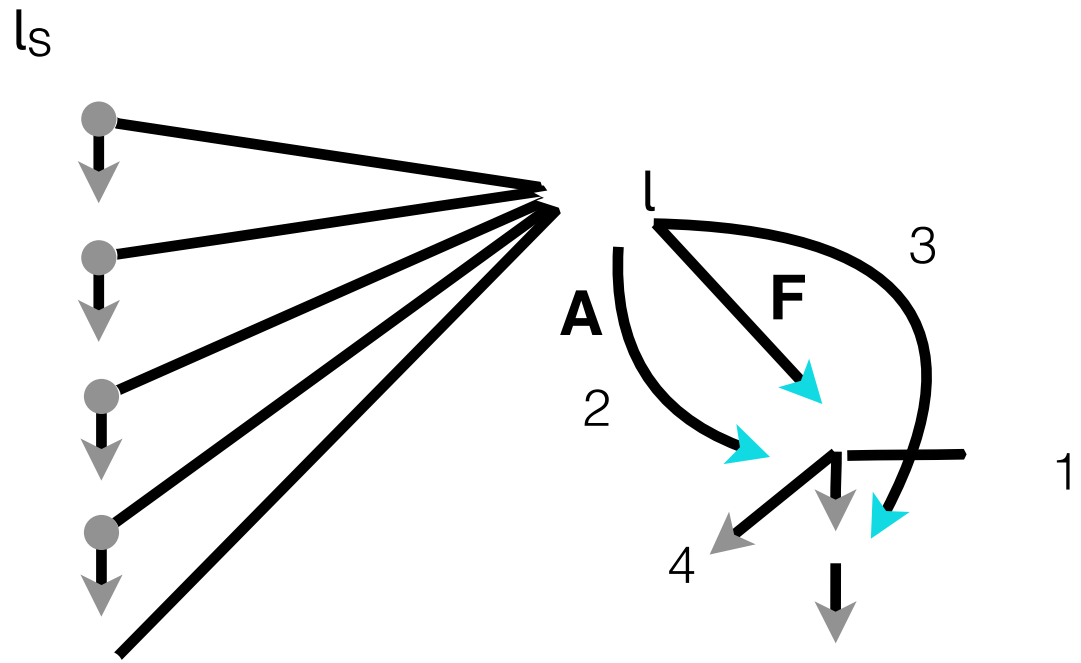
FL 👉 Δ_L ➢ D

A 👉 **F** ➢ D

F 👉 **F**D ➢ D

AD 👉 Δ_D ➢ D

Δ_r 👉 **A** ➢ D



A may be Δ_L or Δ_r but not Δ_D

Access to a variable

Interpretation inside the SECD machine

$$\langle s, e, (\textbf{ld}\ i)\cdot c, d\rangle \rightarrow \langle (\text{get}\ e\ i)\cdot s, e, c, d\rangle$$

Compilation to the DAM machine

$$[\![(\textbf{ld}\ i)]\!]_\iota \quad \rightarrow \quad \text{Alloc}(\iota, \textbf{E}\cdot D^i\cdot L, \textbf{S}, \textbf{S})$$

Closures

Interpretation inside the SECD machine

$$\langle s, e, (\textbf{ldf}\ c_f)\cdot c, d\rangle \rightarrow \langle \langle c_f\ e\rangle \cdot s, e, c, d\rangle$$

Compilation to the DAM machine

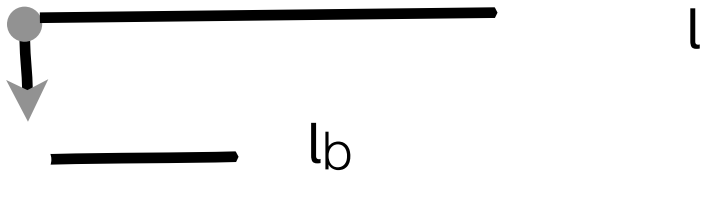
$$[\![(\textbf{ldf}\ c_f)]\!]_\iota \rightarrow \text{movAux}(\iota, [\![c_f]\!]_\iota)$$

- Alloc(ι, **A**, **E**, **A**)
- Alloc(ι, **A**, **S**, **S**)

movAux(ι,ι_b)

A 👉 DL ➢ DD

L 👉 L ➢ D



Application

Interpretation inside the SECD machine

$\langle a \cdot \langle c_f\ e_f \rangle \cdot s,\ e,\ \mathbf{app} \cdot c,\ d \rangle \rightarrow \langle [],\ a \cdot e_f,\ c_f,\ \langle s,e,c \rangle \cdot d \rangle$

Compilation to the DAM machine

$[\![\mathbf{app}]\!]_\iota \rightarrow \mathrm{Alloc}(\iota, \mathbf{S}\mathrm{L}, \mathbf{S}\mathrm{DLD}, \mathbf{A}) \bullet \mathrm{app}(\iota)$

SL 👉 **E** ➢ D

E 👉 **A** ➢ D

SDL 👉 D ➢ **S**DLL

app(ι)

S	E	A

Return

Interpretation inside the SECD machine

<r·s', e', **ret**·c', <s,e,c>·d> → <r·s, e, c, d>

Compilation to the DAM machine

⟦**ret**⟧ι → return(ι)

return(ι)

S	E

E ☚ **S**DL ➢ D

SD ☚ **S**DDD ➢ **S**DDL

Micro coded machine

Many sequences of node are identical

Sharing by putting some code inside the machine

S ≜ …
A ≜ …
F ≜ …
E ≜ …

$[\![(\textbf{ld}\ i)]\!]_\iota \rightarrow \textbf{A} \multimap \textbf{E}\cdot D^i\cdot L \succ \textbf{Ld}$

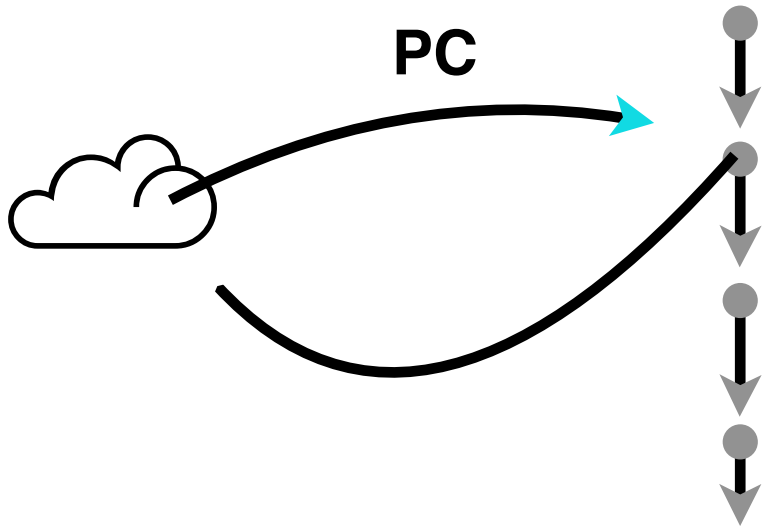
But we have loose the continuation

Ld ≜ …
Ldf ≜ …
App ≜ …
Ret ≜ …

Unlinked code

All the instructions have a static link to the machine

Let's the machine to install itself before running



unlinked code

Micro-sequenser

Install the machine and goto the instruction

(Fetch) **PC**L ☚ <m> ➢ D

K ☚ **PC**D ➢ **PC**

User instruction

A ☚ **E**·D^{i}·L ➢ **Ld**

Micro code

(Ld) Alloc(ι, **E**·D^{i}·L, **S**, **S**) • (L ☚ L ➢ **Next**)

Epilogue

(Next) **PC**L ☚ **PC** ➢ D

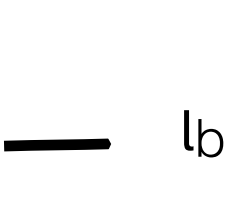
PC ☚ **K** ➢ **Fetch**

Closures

User instruction

A 👉 DL ➢ **Ldf**

L 👉 L ➢ D



Micro code

(Ldf)

Alloc(<m>, **A**, **E**, **A**)

- Alloc(<m>, **A**, **S**, **S**)
- (**K** 👉 **K**D ➢ **Next**)

Application

User instruction

L 👉 L ➢ **App**

Micro code

Alloc(<m>, **S**L, **S**DLD, **A**)

(App)

- **S**L 👉 **E** ➢ D
- **E** 👉 **A** ➢ D
- **K** 👉 **S**DLL ➢ D
- **S**DL 👉 **PC**D ➢ **Next**

Return

User instruction

L 👉 L ➢ **Ret**

Micro code

E 👉 **S**DL ➢ D

(Ret)

- **K** 👉 **S**DLL ➢ D
- **S**D 👉 **S**DDD ➢ **Next**

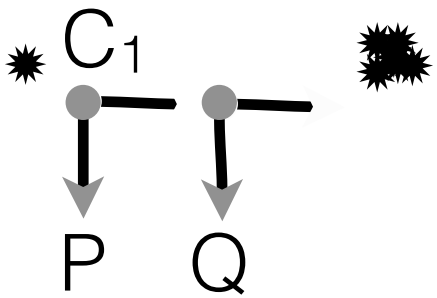
Exercice

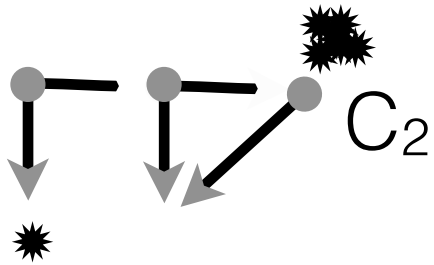
Do the AppRet micro-code
compiled for a tail call

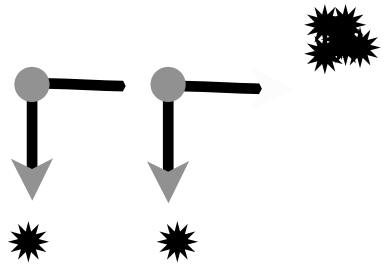
Synchronisation patterns

Parallelism = token generation

$[\![(P \parallel Q)]\!] \rightarrow par([\![P]\!], [\![Q]\!])$

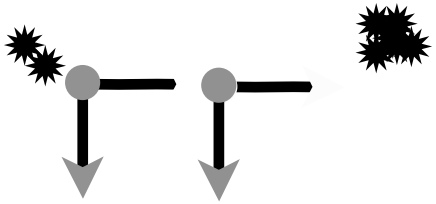


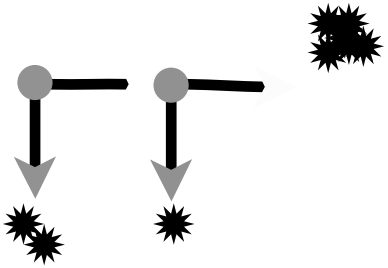




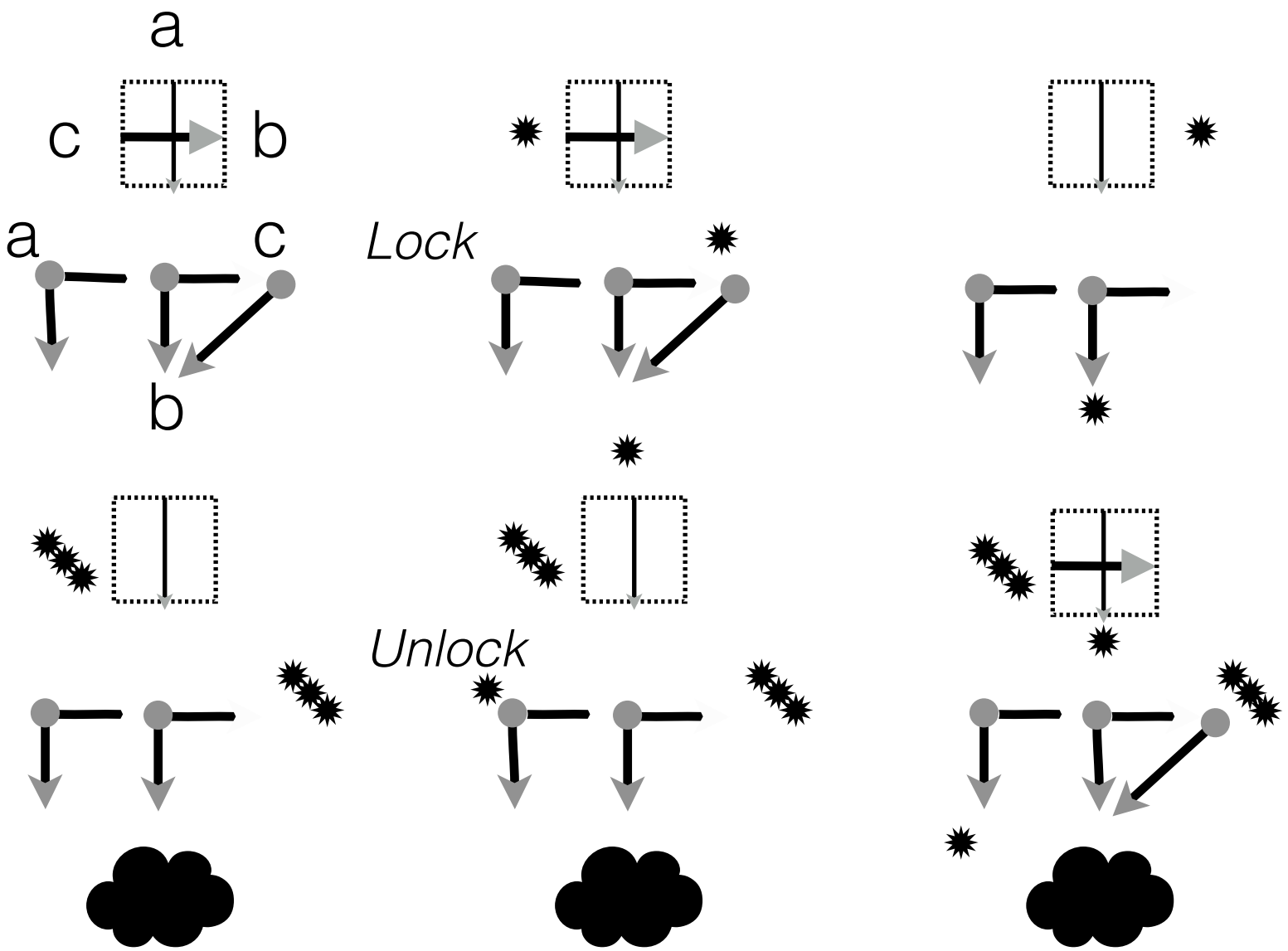
$C_1 \triangleq LLD \multimap LD \succ D$ $C_2 \triangleq D \multimap L \succ D$

Must be synchronised



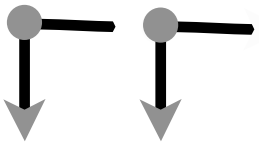


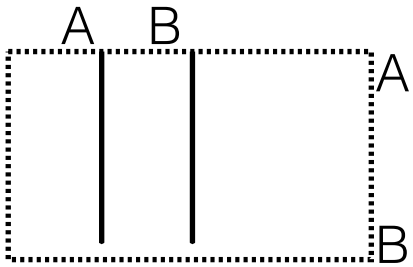
mutex = token retention



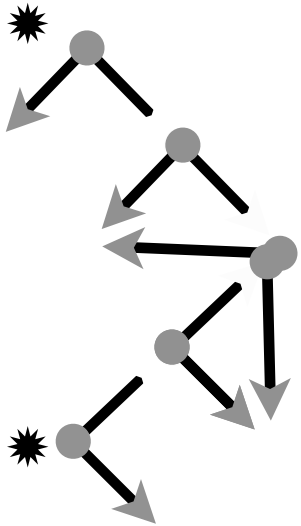
Multiple entries mutex

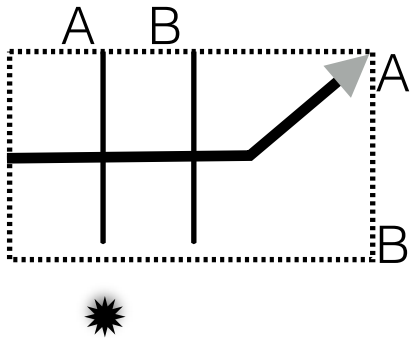
1 entry

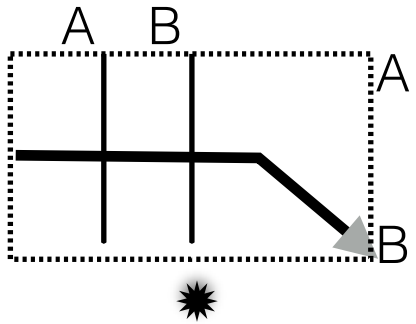




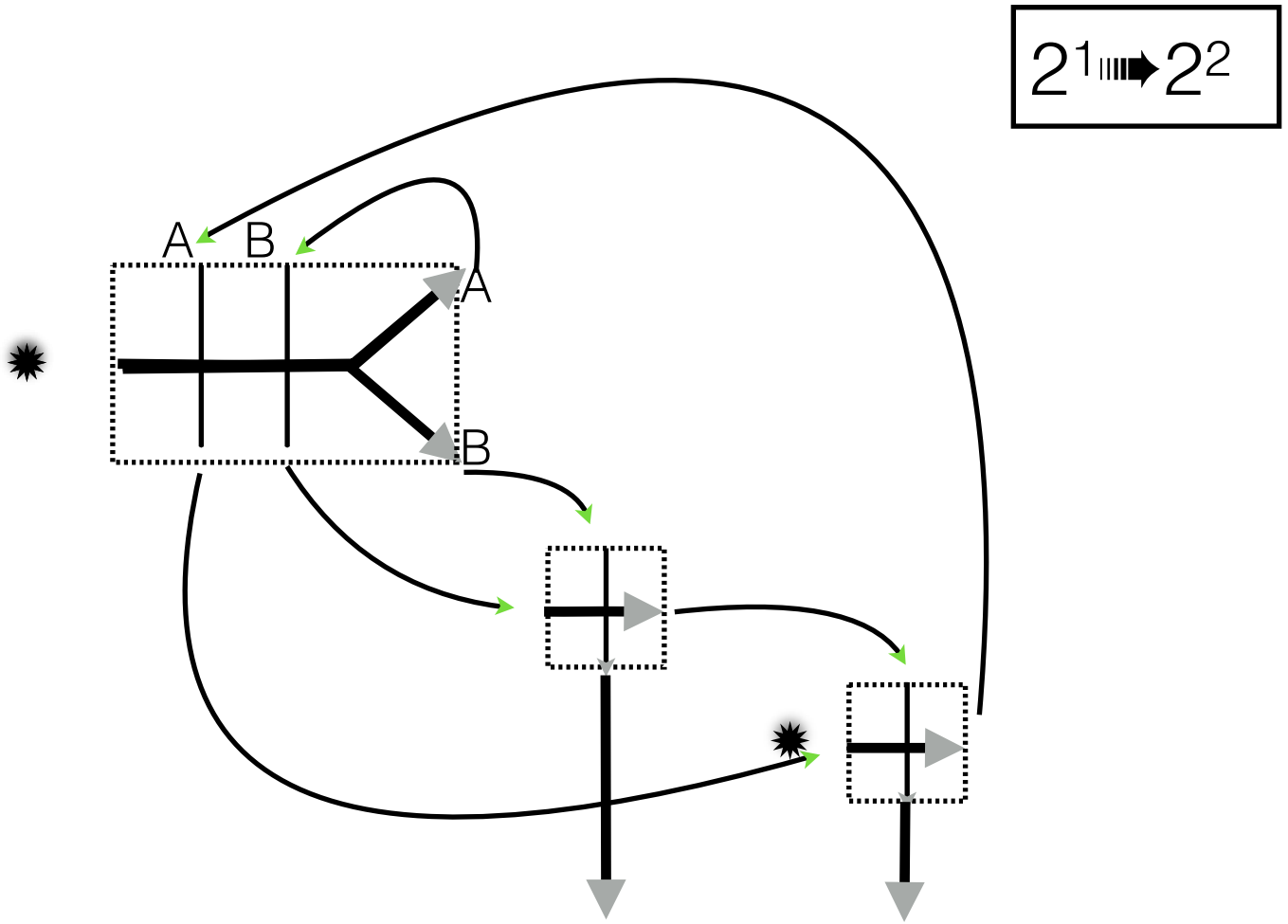
2 entries



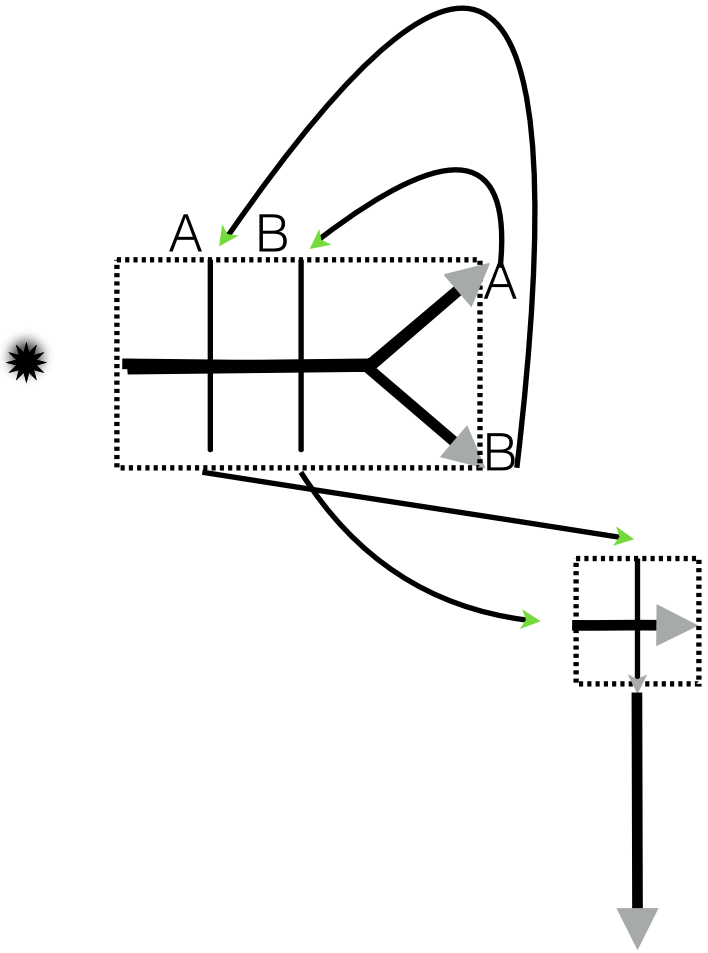




Waiting 2 tokens on same node

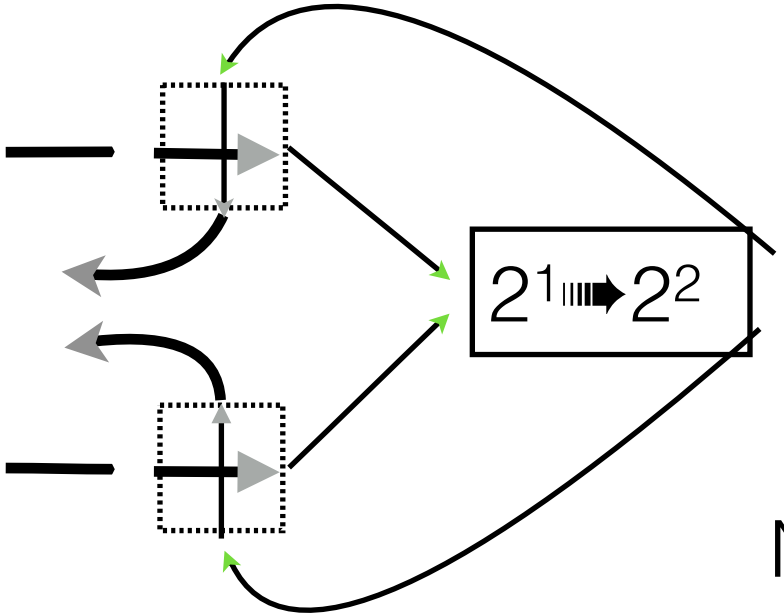


Bad implementation of $2^1 \Rightarrow 2^2$



Synchronisation

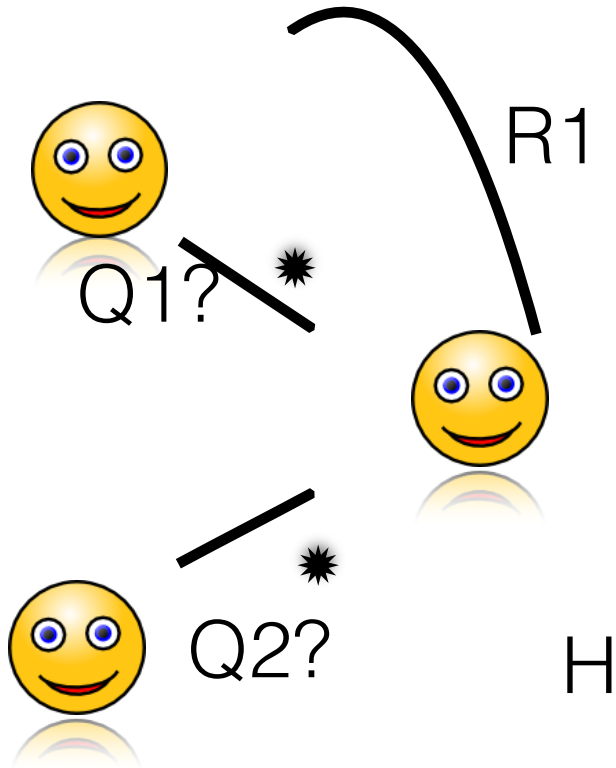
Waiting 2 tokens on two nodes = $2^2 \Rightarrow 2^2$



Not enough for CCS : the continuations are static

Call convention

Concurrent call convention



Callers must give their arguments at a specific place of the callee

Cannot lock a mutex and write a data at the same node (2 writes)

Having values associated to tokens is a solution. Is it essential ?

Wrong protocol

Caller Side

1 Me.Cont = @4
2 Callee.Caller = Me
3 Send the token to Callee
4 *Me.Resp is OK*

1’ Me’.Cont = @4’
2’ Callee.Caller = Me’
3’ Send the token to Callee
4’ *Me’.Resp is OK*

Callee Side

1 Close the door
2 Caller.Resp = f(Caller)
3 Give a token to Caller.Cont
4 Open the door
5 Stop

2 and 2’ are concurrent writes

spinlock / polling ?

1 Me.Run = true
2 Me.Cont = @7
3 Callee.Caller = Me
4 Give a token to Callee
5 While Me.Run
    Callee.Caller = Me
6 Stop
7 *Me.Resp is OK*

1 Close
2 C=Caller
3 C.Run = false
4 C.Resp = f(C)
5 Open > C.Cont

After Callee@5, Callee.Caller may still have Me instead of Me' (for 2 reasons)

5' While Me'.Run
    Callee.Caller = Me'

Waiting caller inactivity

1 Me.Wait = @4
2 Callee.Caller = Me
3 Give a token to Callee
4 Callee.Caller = Me > Me.Wait
5 > Callee@4
6 *Me.Resp is OK*

1 Close
2 C=Caller
3 C.Wait = C@5 > Stop
4 C.Resp = f(C)
5 Open > C@6

Waiting to caller's value

1 Me.Wait = @3
2 Give a token to Callee
3 Callee.Caller = Me > Me.Wait
4 > Callee@5
5 *Me.Resp is OK*

1 Close
2 Caller==None ? 2 : 3
3 C = Caller
4 C.Wait = C@4 > Stop
5 Caller=None
6 C.Resp = f(C)
7 Open > C@5

The first write in the caller can be removed

None equality

All callers but None

0 > Callee@3
1 …

1 Close
2 > Caller@0
3 C = Caller
4 …

None

0 > Callee@2

Can we replace 2&3 with C = Caller > Caller@0 ?

Asynchronous Semantics

Incorporating the path following in the rules

New structure

$\mu \in \text{Mach} \triangleq \text{Label} \rightarrow \text{Node}$

$\text{Node} \triangleq \textit{Tokens}\text{:Nat} \times \textit{L}\text{:Label} \times \textit{D}\text{:Label} \times \textit{Code}\text{:Instr}$
$\times\ \textit{Values}\text{: (Path x Label)}^3$
$\times\ \textit{SetJumps}\text{: } \mathcal{P}\text{(Edge x Label x Label)}$

Values will contain the intermediate values of the 3 follow ($\mathbb{F}$) calls in the sequential semantics

Setjumps is a set of pending affectation that a node have to perform on one of its fields (denoted by the edge), the second label point to the node which have to receive a token

Start

$$\mu(\iota)=n \quad \wedge \quad n.\mathit{Tokens}=t+1 \quad \wedge \quad n.\mathrm{SetJumps}=\emptyset$$

$$\wedge\ n.\mathrm{Code} = \Delta_d \cdot \delta \twoheadrightarrow \Delta_s \succ \Delta_n \quad \wedge \quad n.\mathrm{Values} = ()$$

$$\mu \longrightarrow \mu[\iota \leftarrow n[\mathit{Tokens} \leftarrow t][\mathrm{Values} \leftarrow ((\Delta_d,\iota)(\Delta_s,\iota)(\Delta_t,\iota))]$$

A node can start its execution only if it have at least one token, it have no pending writes and if it doesn't currently executing its instruction

To start, the node have only decrease the number of its tokens and initiate its *Values* field

No communication

Following one edge

$$\mu(\iota)=n \;\wedge\; n.Values=(\ldots(\delta\cdot\Delta,\iota')\ldots) \;\wedge\; \mu(\iota')=r \wedge\; r.\delta=\iota''$$

$$\mu \longrightarrow \mu[\iota \leftarrow n[Values \leftarrow (\ldots(\Delta,\iota'')\ldots)]$$

Field can be fetched with a message passing style

SendWrite

$$\mu(\iota)=n \ \wedge \ n.\mathit{Values}=((\varepsilon,\iota_d)(\varepsilon,\iota_s)(\varepsilon,\iota_t)) \wedge \mathbf{\iota_d <> \iota}$$

$$\wedge\ n.\mathit{Code} = \Delta_d{\cdot}\delta\!\!\rightarrow\!\Delta_s\!\!\succ\!\Delta_n \quad \wedge\ \mu(\iota_d)=n_d$$

$$\mu \longrightarrow \mu[\iota \leftarrow n[\mathit{Values}:=()]]$$
$$[\iota_d \leftarrow n_d[\mathit{SetJumps} \leftarrow n_d.\mathit{SetJumps} \cup \{(\delta,\iota_s,\iota_t)\}]]$$

When the path resolution is finish, the node transmit a message to the node which have to change its field. The label of the node which have to receive the token is included in the message

The token must be valid only after affectation

A node cannot send a message to itself

SetJump

$$\frac{\mu(\iota)=n \wedge n.\mathit{SetJumps}=\{\delta,\iota_s,\iota_t\} \cup W \wedge n.\mathit{Values} = \emptyset}{\mu \longrightarrow \mathrm{Jump}(\iota_t,\mu[\iota \leftarrow n[\delta \leftarrow \iota_s][\mathit{SetJumps} \leftarrow W]])}$$

$$\mathrm{Jump}(\iota,\mu)= \mu[\iota=\mu(\iota)[\mathit{Tokens} = 1+\mu(\iota).\mathit{Tokens}]]$$

Jump increments the number of token of one specific node

A node can change one of its L or D fields only if it is not executing its instruction

LocalWrite

$$\mu(\iota)=n \quad \wedge \quad n.\mathit{Values}=((\varepsilon,\iota)(\varepsilon,\iota_s)(\varepsilon,\iota_t))$$

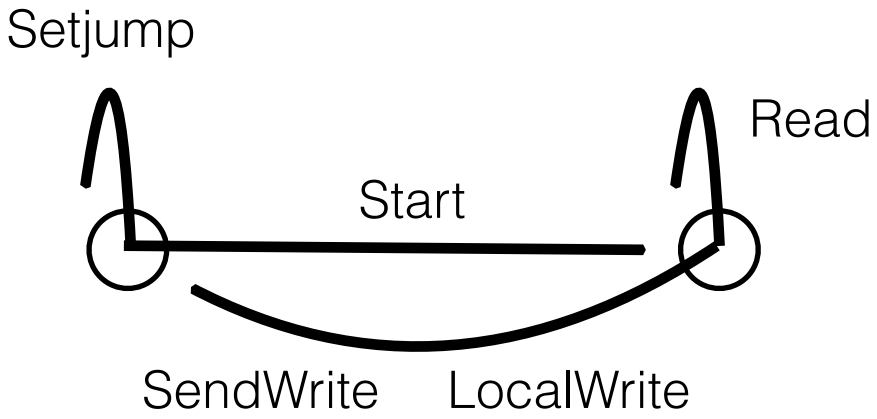
$$\wedge\ n.\mathrm{Code} = \Delta_d \cdot \delta \!\!\rightarrow\!\! \Delta_s \!\!\succ\!\! \Delta_n$$

$$\mu \longrightarrow \mathrm{Jump}(\iota_t, \mu[\iota \leftarrow n[\delta \leftarrow \iota_s][\mathit{Values} := ()]])$$

A local write must be done before all pending writes

Without the LocalWrite rule, mutex and token generation doesn't work

Time life of a node



The minimal instance

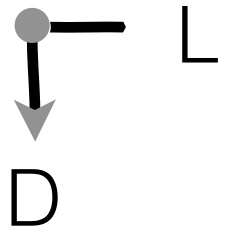
Could we bound the number of instructions ?

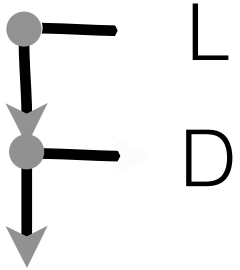
Simulating the path following by the machine itself

Get extra memory

Add a node on every dark edge and compile the code on the fresh node

$\Delta_d \cdot \delta \twoheadrightarrow \Delta_s \succ \Delta_n$





$[\![T(\Delta_d \cdot \delta) \twoheadrightarrow T(\Delta_s) \succ T(\Delta_n)]\!]_l$

All paths must be adapted due to the inserted nodes

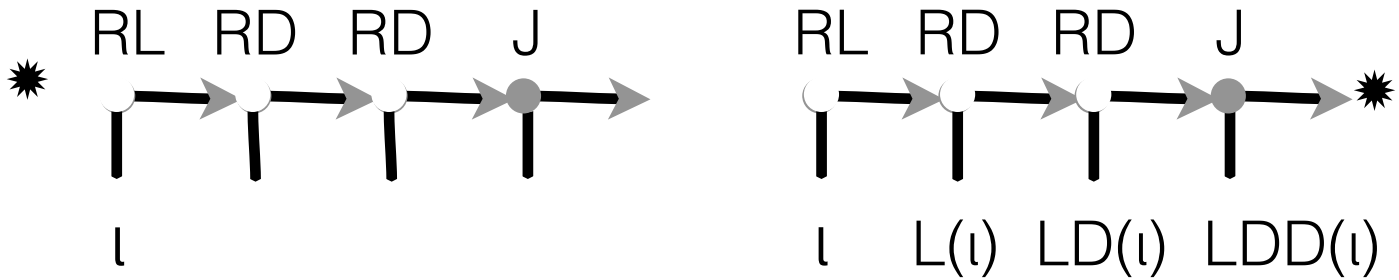
$T(\varepsilon)=\varepsilon$ $T(L \cdot \Delta)=L \cdot T(\Delta)$ $T(D \cdot \Delta)=DL \cdot T(\Delta)$

Follow the paths

$$[\![\Delta_d \cdot \delta \text{👉} \Delta_s \text{➢} \Delta_n]\!]_\iota = \mathrm{Comp}(P(\iota,\Delta_d)\ ,P(\iota,\Delta_s)\ ,P(\iota,\Delta_n))$$

Read each field and transmit the result to continuation

Δ=LDD



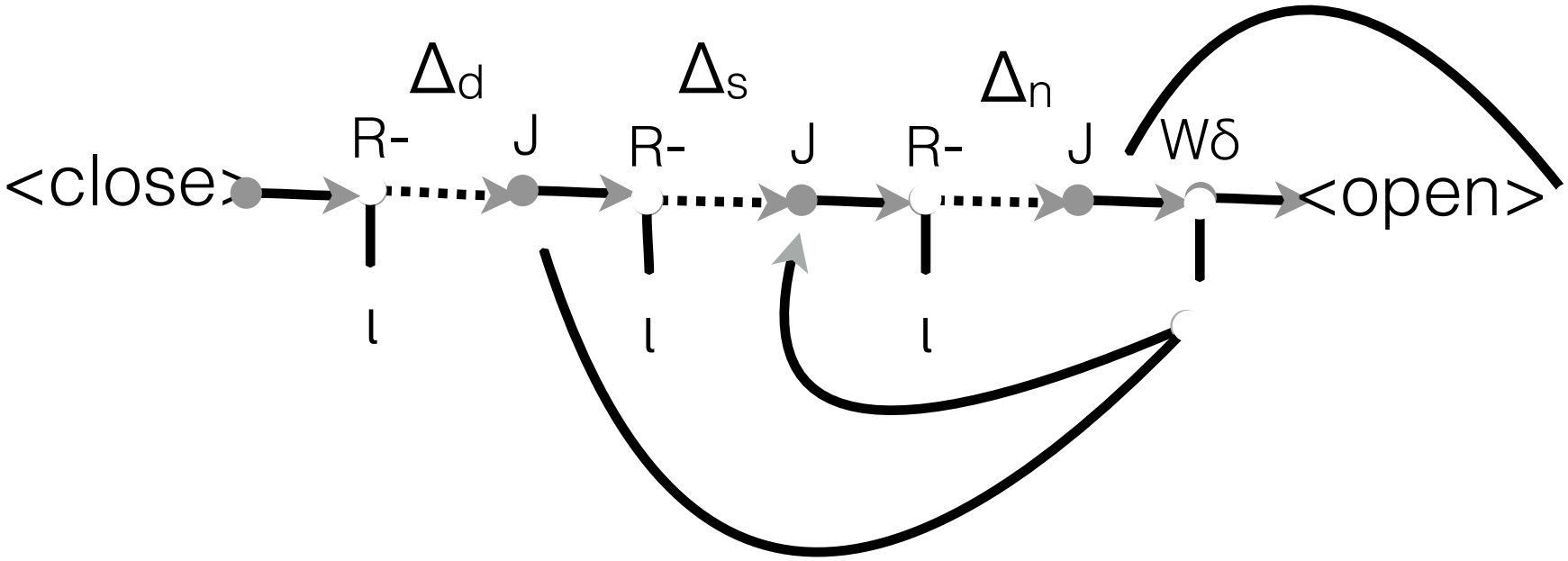
RD ≜ DL👉 LD➢ D RL ≜ DL👉 LL➢ D J ≜ L👉 L➢ D

$R(\delta \cdot \Delta) = \text{`}R\delta\text{'} \bullet R(\Delta)$  $R(\varepsilon) = \text{`}J\text{'}$

$P(\iota,R(LDD))$ $P(\iota,\Delta_d) \triangleq \mathrm{SetLight}(\iota,R(\Delta_d))$

Finishing the job

$\Delta_d \cdot \delta \twoheadrightarrow \Delta_s \succ \Delta_n$

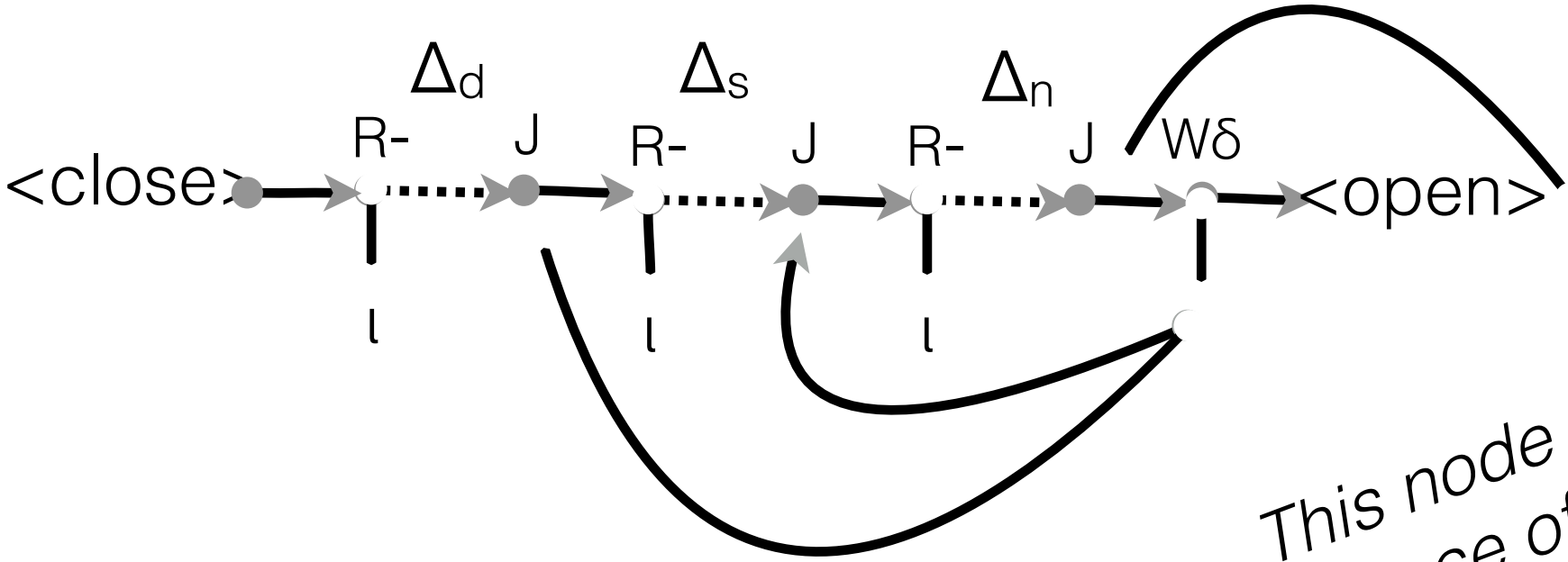


WL ≜ LLLL ↠ LD ≻ D

WD ≜ LLLD ↠ LD ≻ D

Reducing path length

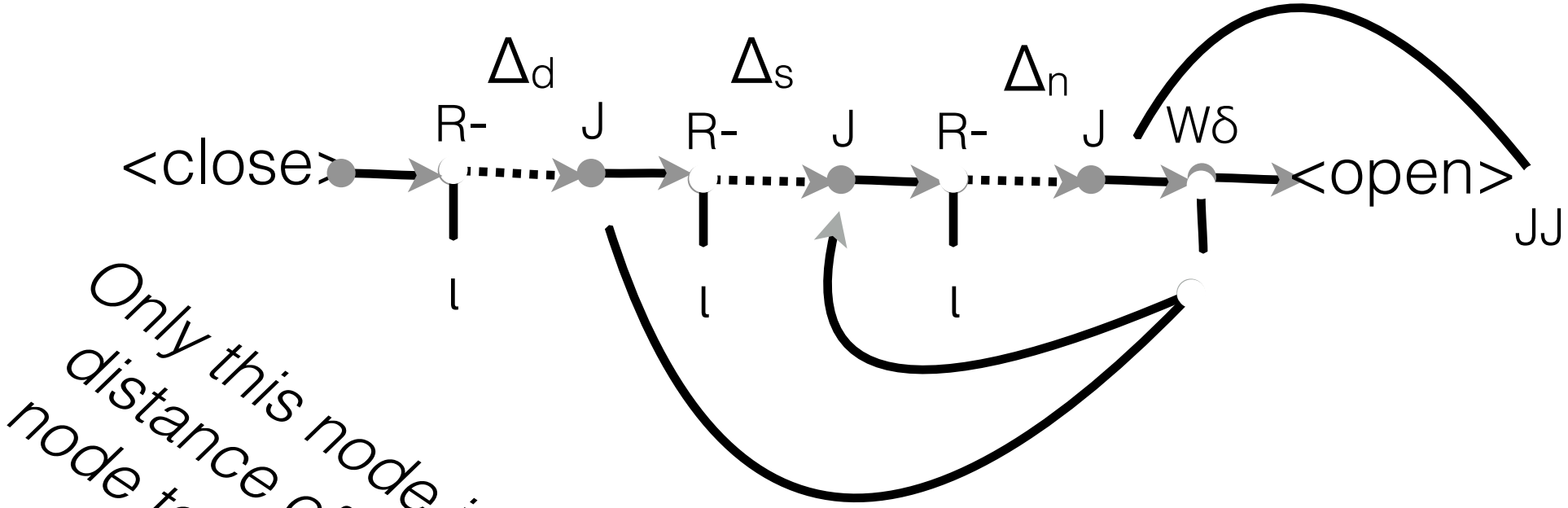
Reducing to paths of length 3 is easy



This node is at a distance of 2 to the node to be modified

Reducing path length

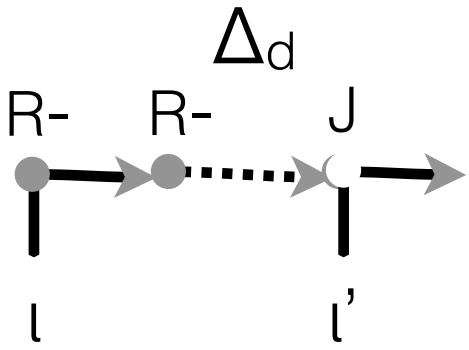
Reducing to paths of length 2 is challenging

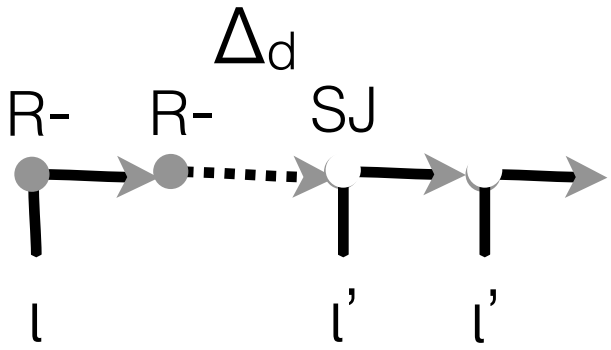


A fresh place for an instruction

Locally unused instruction

SJ ≜ DL 👉 L ➢ DD



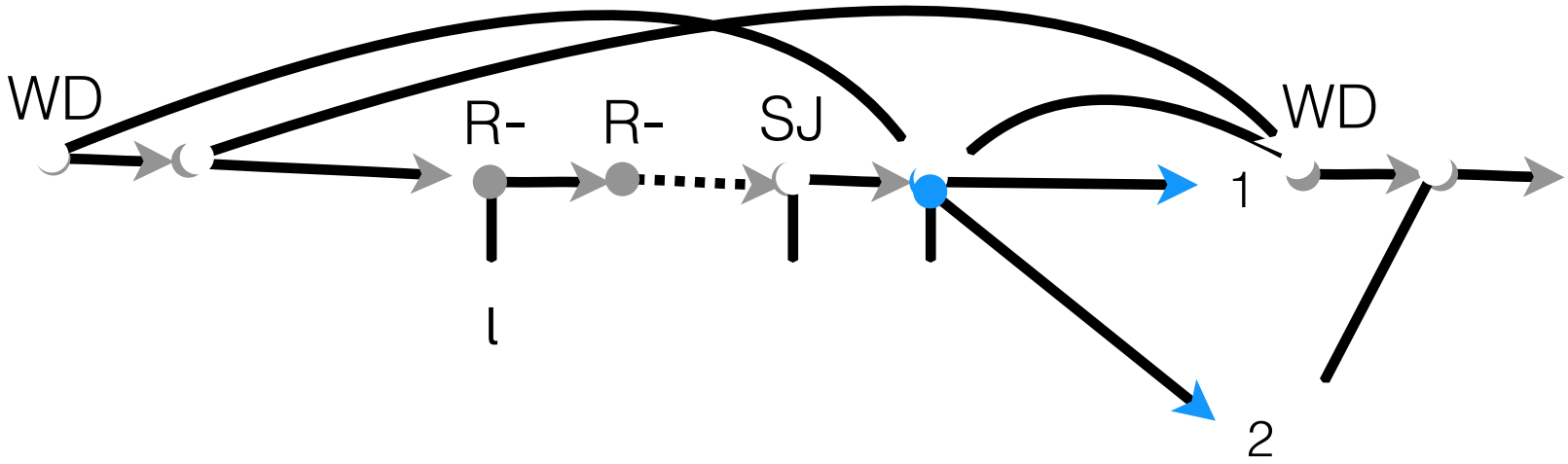


ι' is the label of the node to be modified

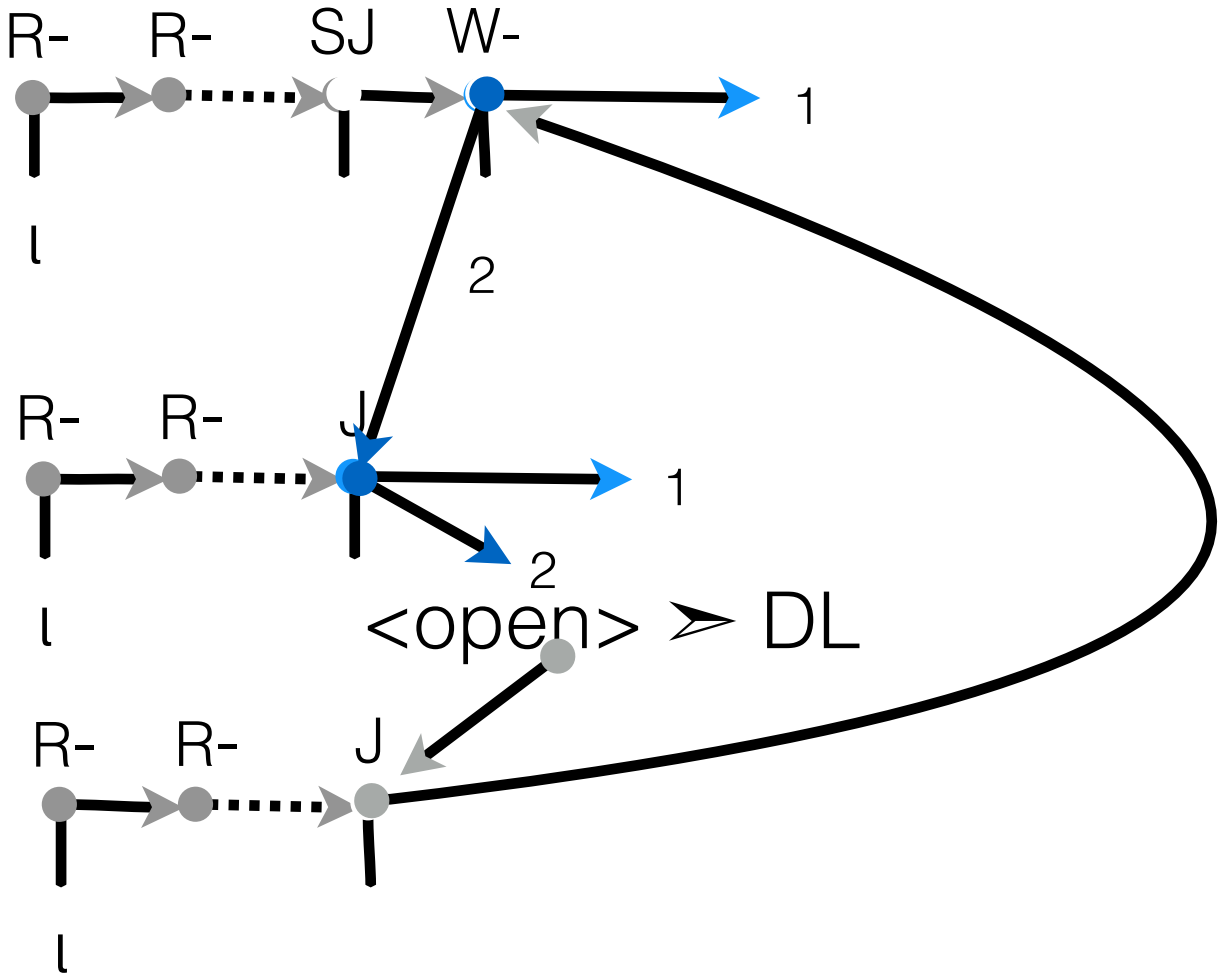
After path resolution this link can be modified

Edge modification afterward

WD ≜ LD ☚ DL ➢ D



Going throw the modified path



Minimal? set of instructions

RL ≜ DL 👉 LL ➢ D

RD ≜ DL 👉 LD ➢ D

WL ≜ LL 👉 DL ➢ D

WD ≜ LD 👉 DL ➢ D

SJ ≜ DL 👉 L ➢ DD

J ≜ L 👉 L ➢ D

Open ≜ LLD 👉 DL ➢ DL

Close ≜ D 👉 L ➢ D