Madcloud

Distributed State Machine Madness

Jacoby Thwaites, Google

What we'll cover in 30mins

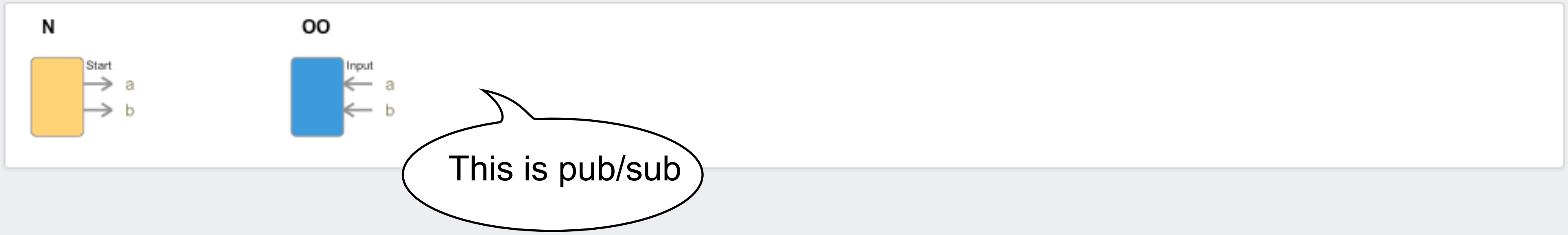
1. The idea
2. Some examples
3. An implementation

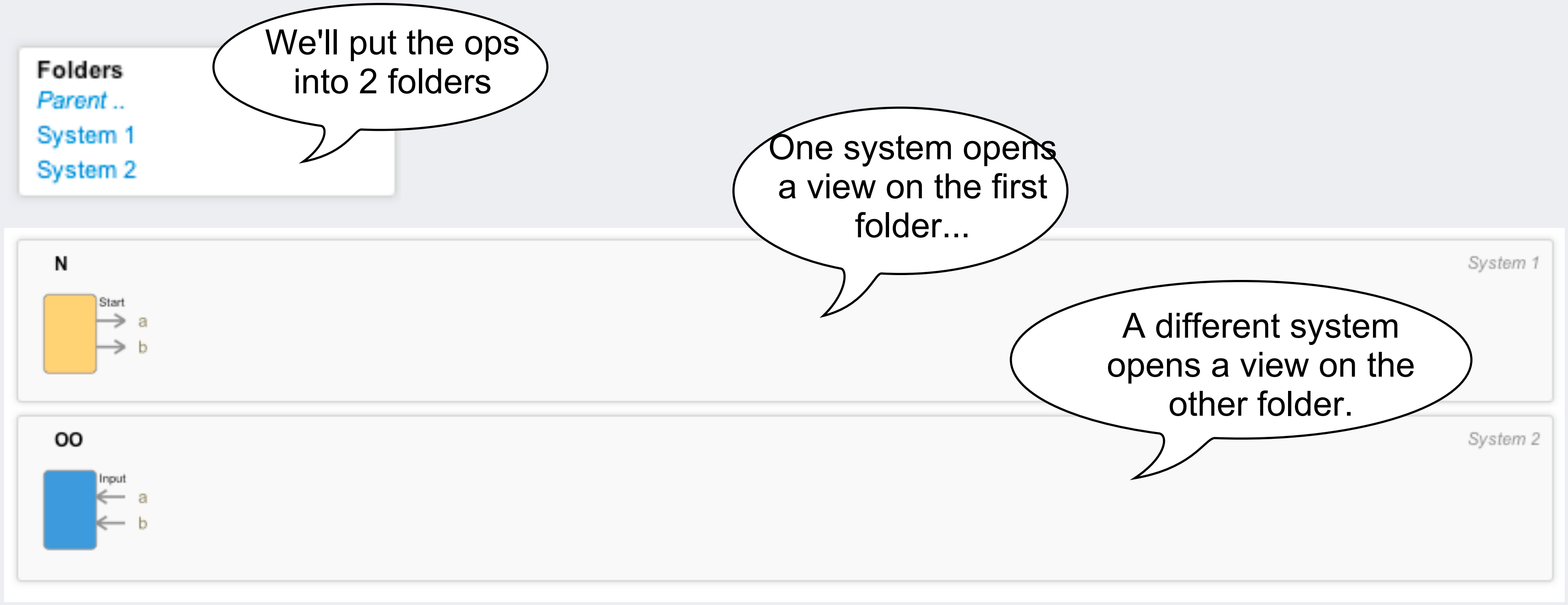
The idea

- A declarative **call-out infrastructure**
  - The inside-out API
    - Everything is a server
    - Nothing is a client
  - The infrastructure maintains process state

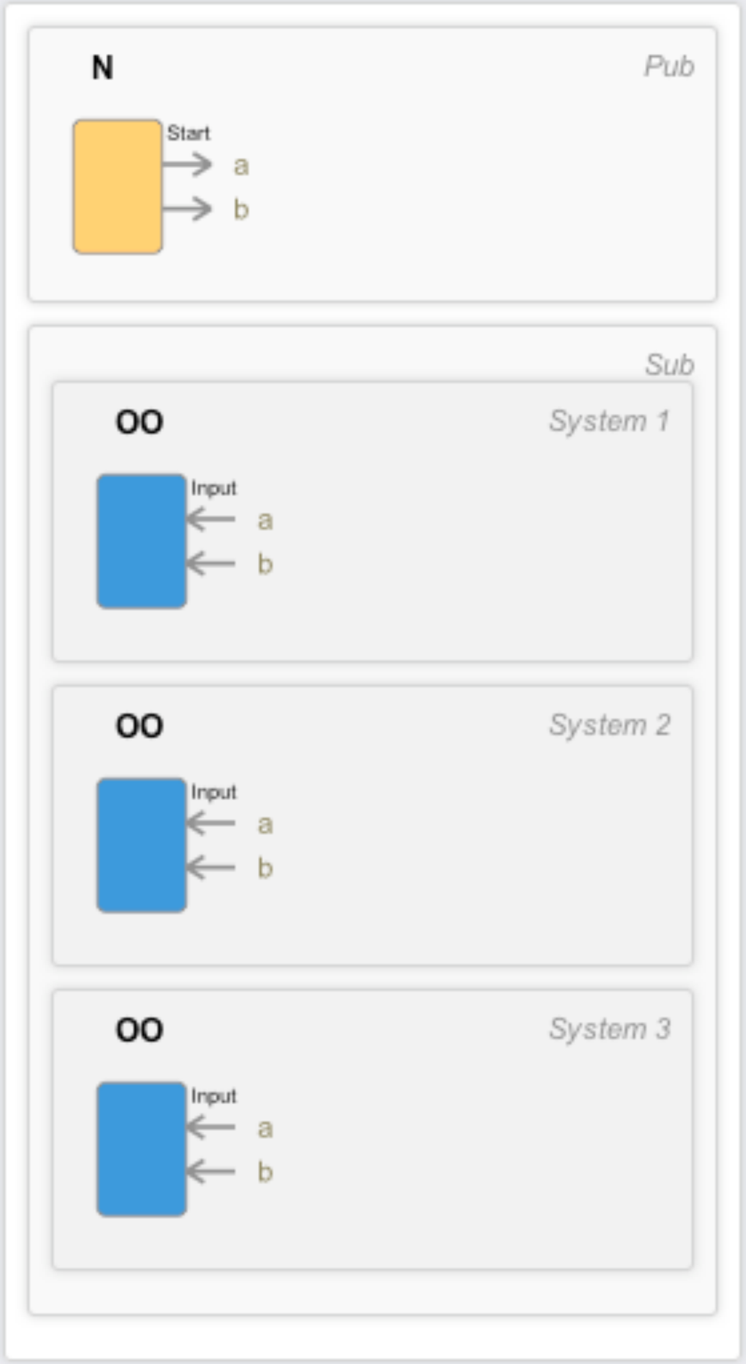
- Purpose
  - Mutually ignorant services combine to form applications
    - A bit like BPEL
- Eg
  - Some businesses who don't know each other collaborate to provide a service for a consumer they don't know

Example #1

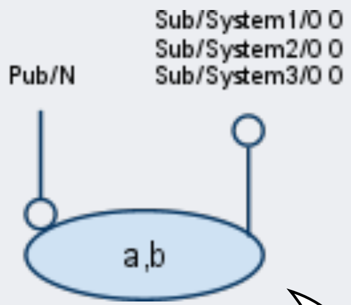

N
Start
a
b
OO
Input
a
b
This is pub/sub


Folders
Parent ..
System 1
System 2
We'll put the ops into 2 folders
One system opens a view on the first folder...
A different system opens a view on the other folder.
N
Start
a
b
System 1
OO
Input
a
b
System 2

Example #1

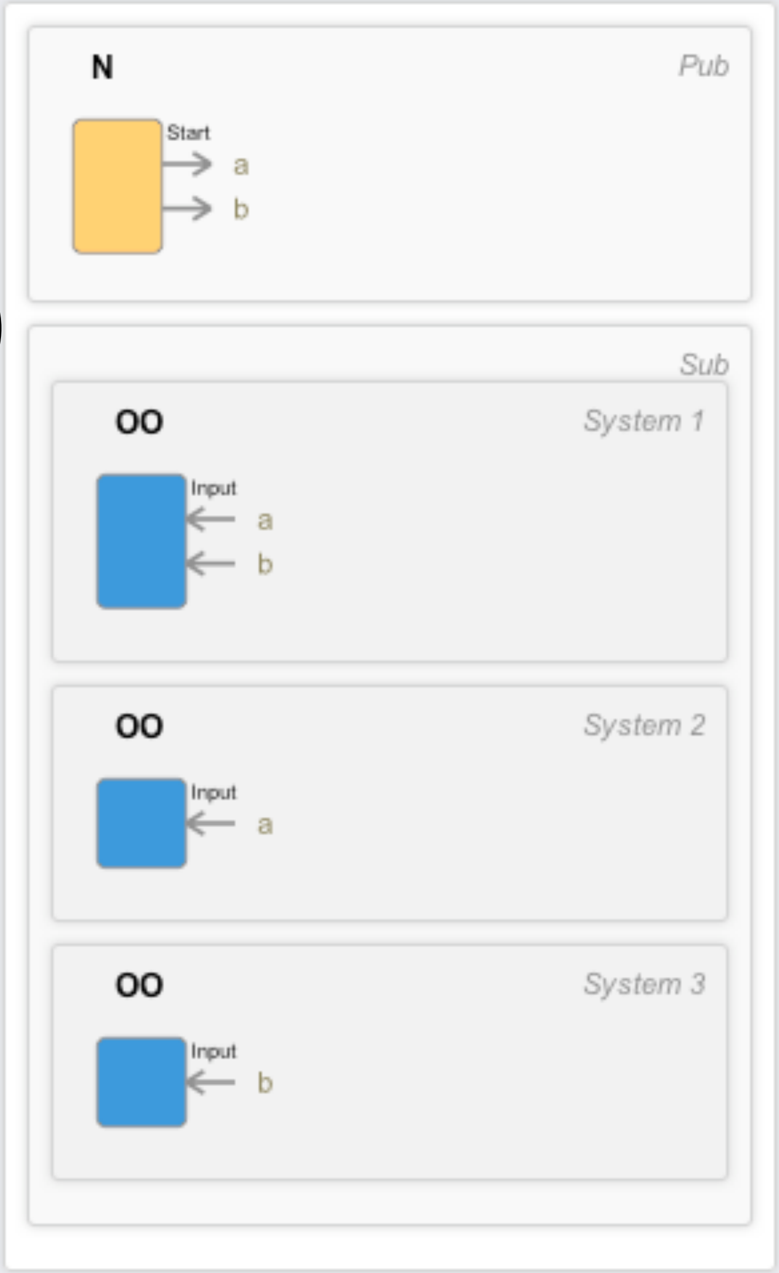


This is like pub/sub topics

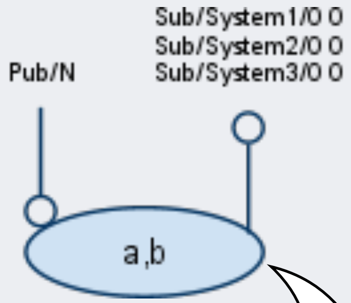


The fieldset state digraph is derived from the unordered set of operations

Example #1

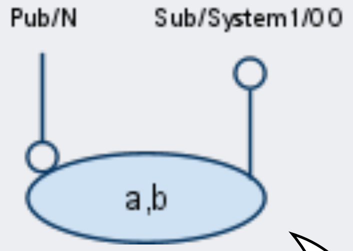


Different systems can have different inputs

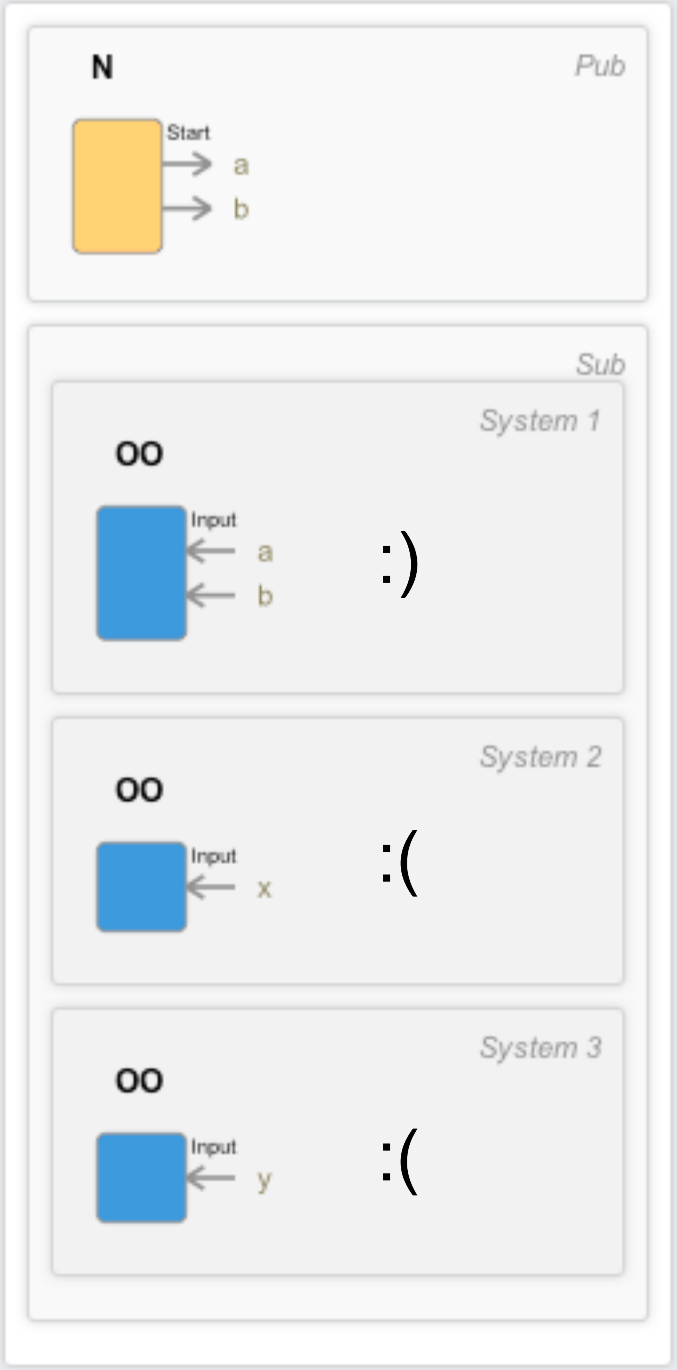


A single fieldset state can supply any op whose input is a subset

Example #1

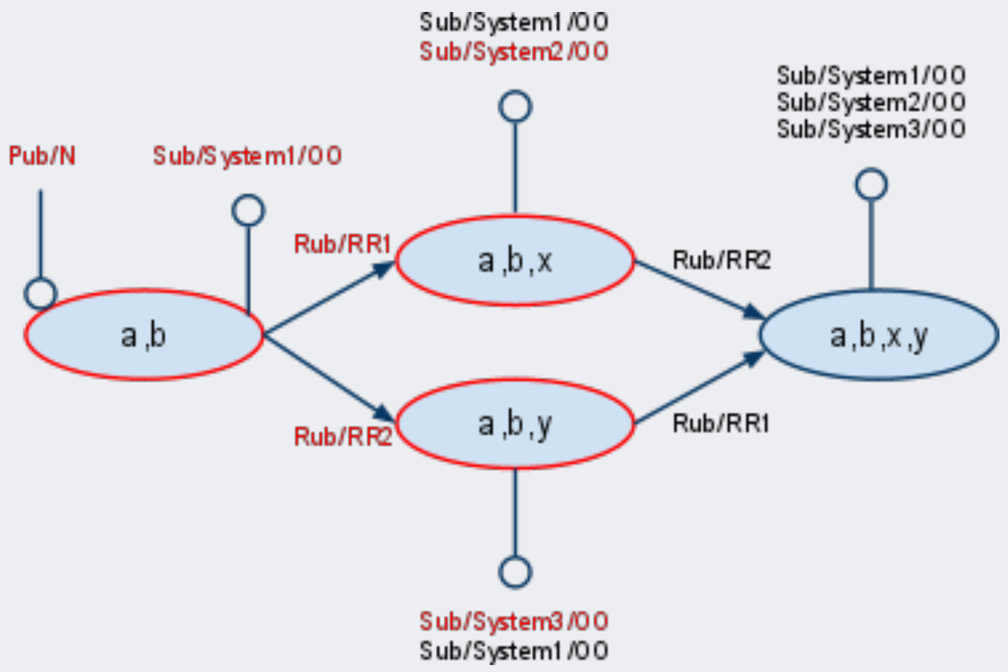


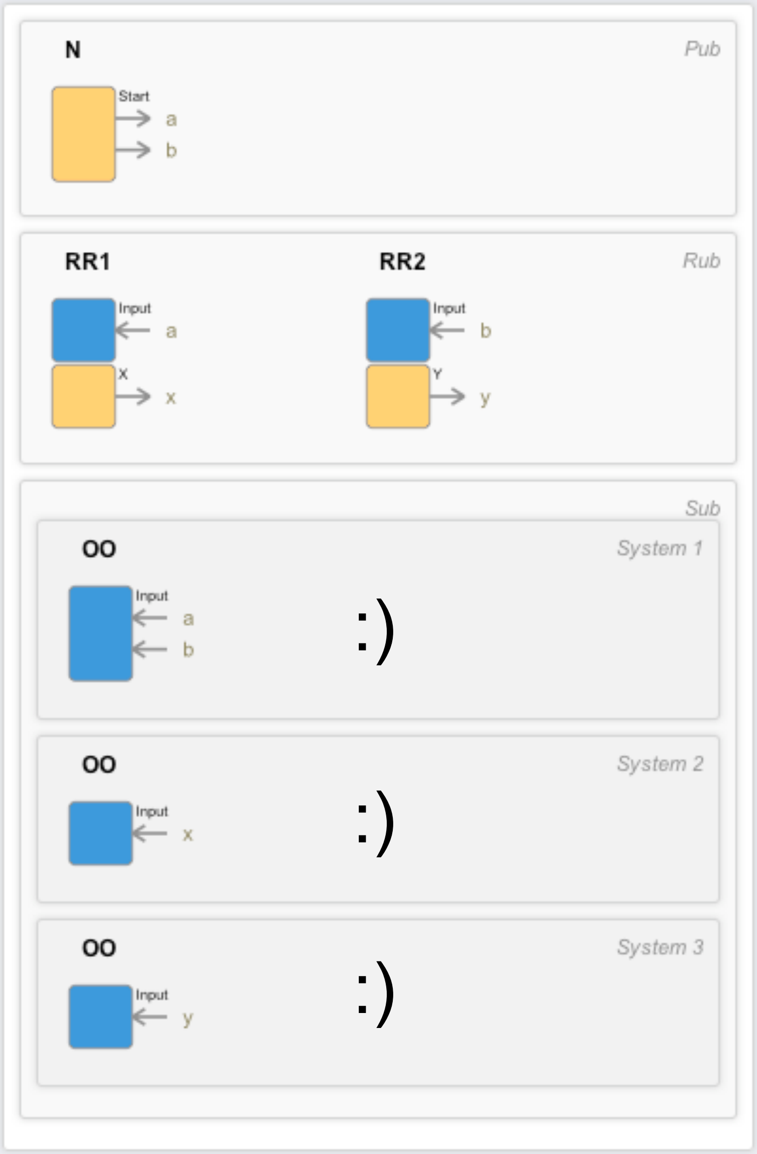
A fieldset state can't supply an op's input if it doesn't have it.



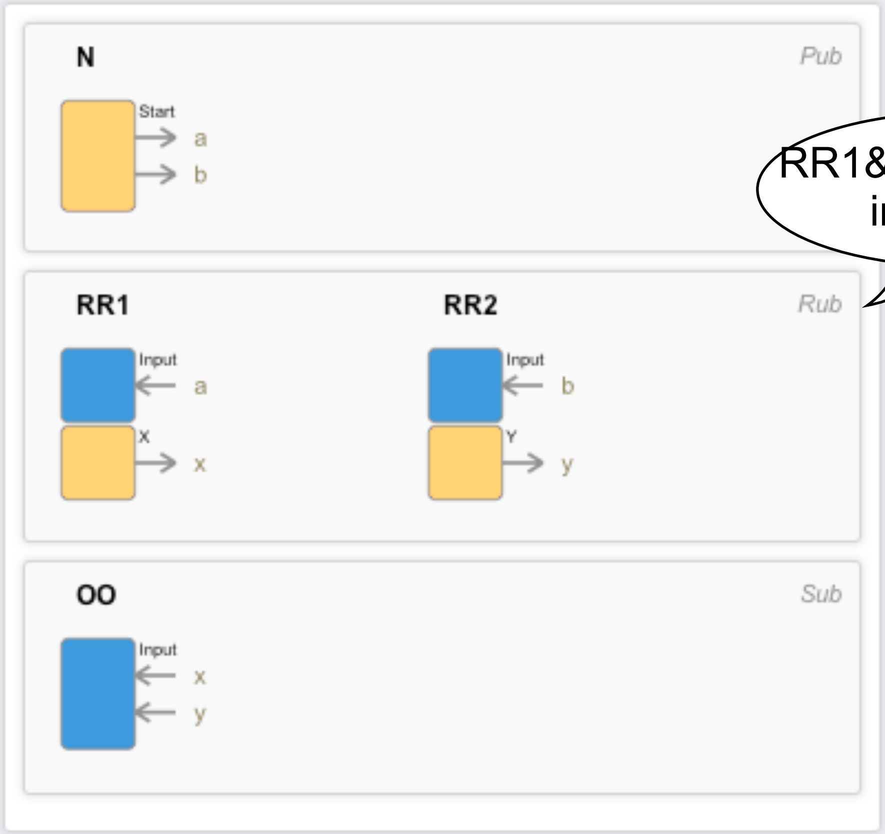
Example #1

Any one fieldset can supply any one op exactly once, during its lifetime



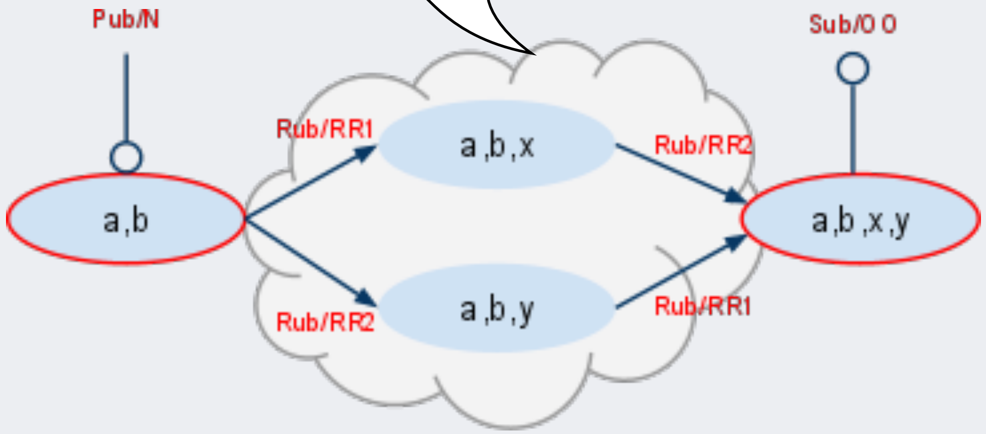


Example #1



RR1& RR2 happen in parallel

Probability cloud. Could be either sequenced, or in parallel



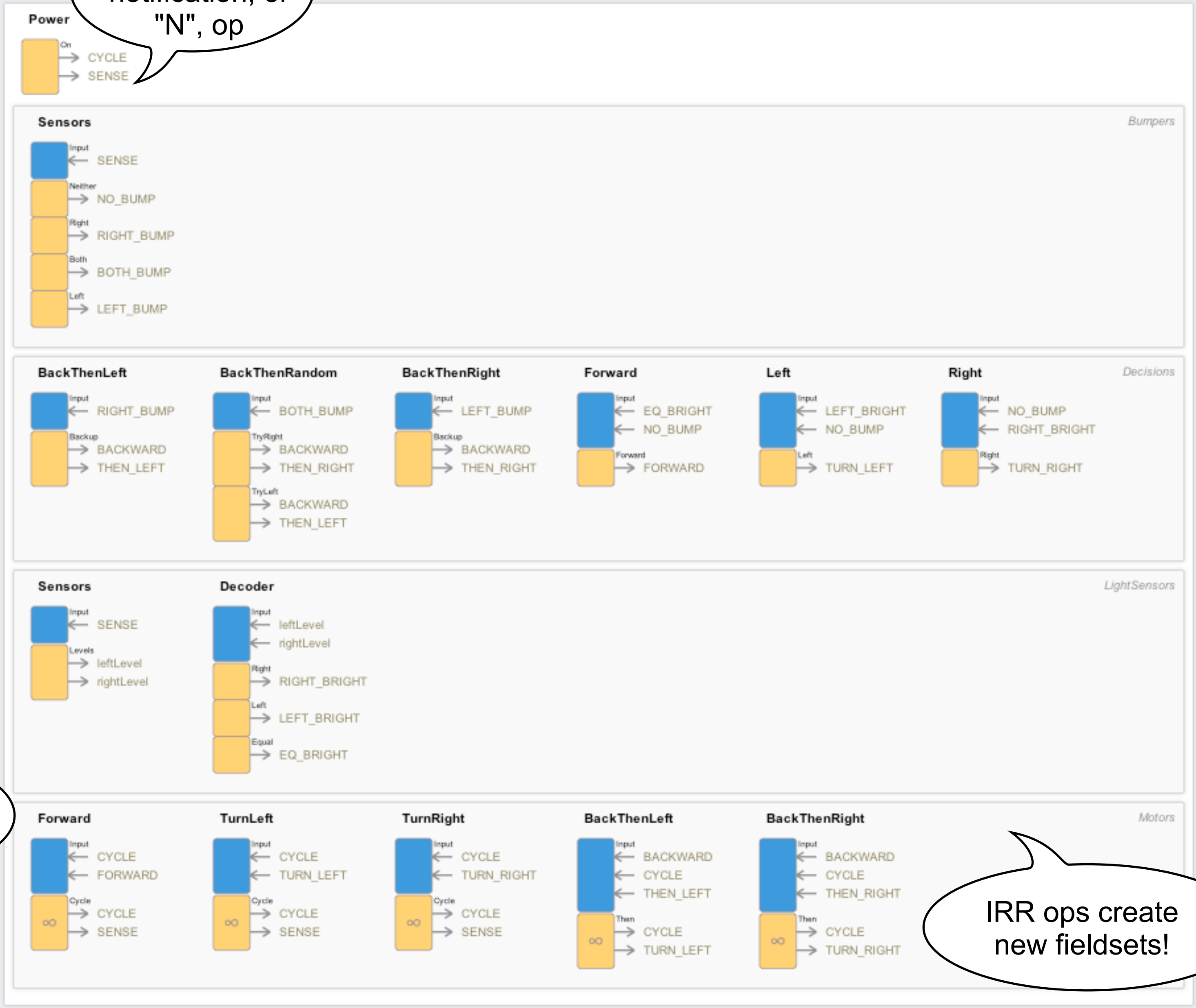
Example #2 is a pub/rub/sub robot

This is a notification, or "N", op

This is a request/reply, or "RR" op

These are iterative request/reply, or "IRR" ops

IRR ops create new fieldsets!

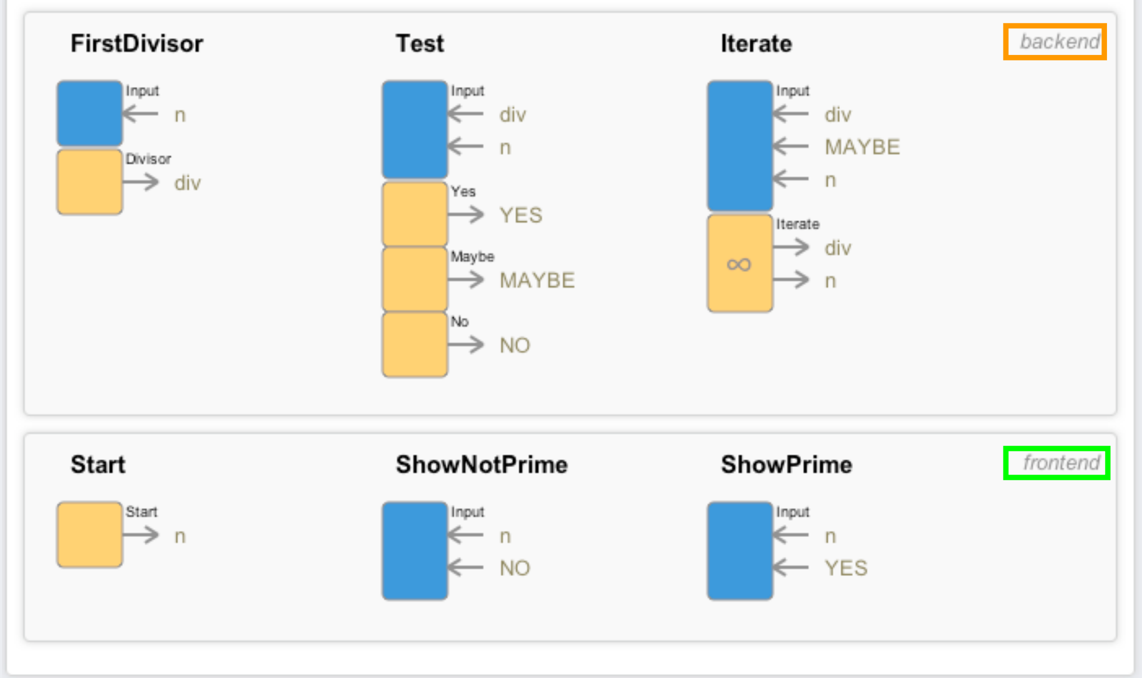


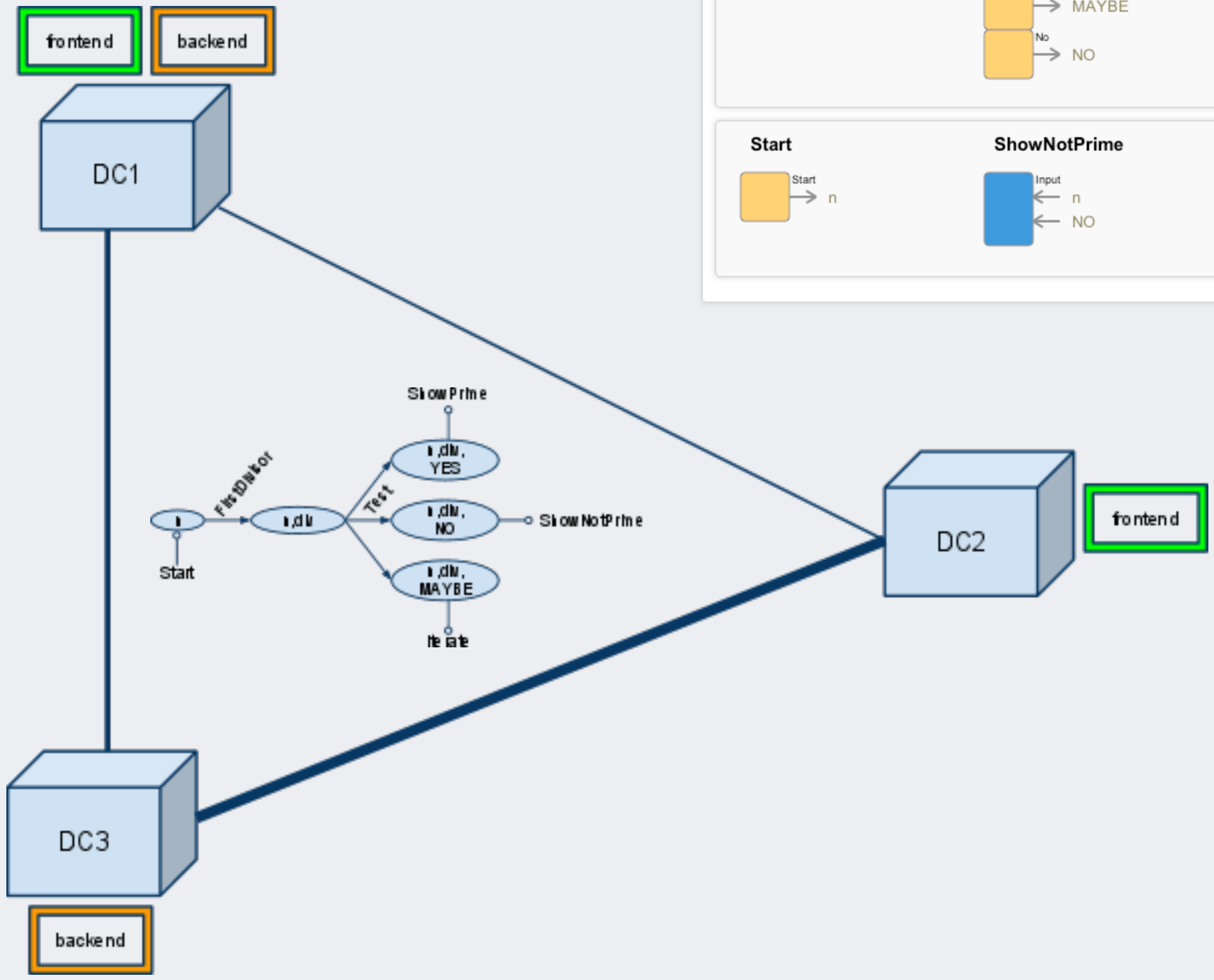
Pause before Distributed State

Distributed state

1. A **process** means everything that happens because of a single Notification operation.
2. The **process state** is the set of fieldsets in existence at any one instant.
3. Each fieldset has a **fieldset state** at any one instant.
4. Fieldsets are **independent**.
   - So we can put the fieldsets in different datacenters
5. Ergo the state of a process is scattered across those datacenters.

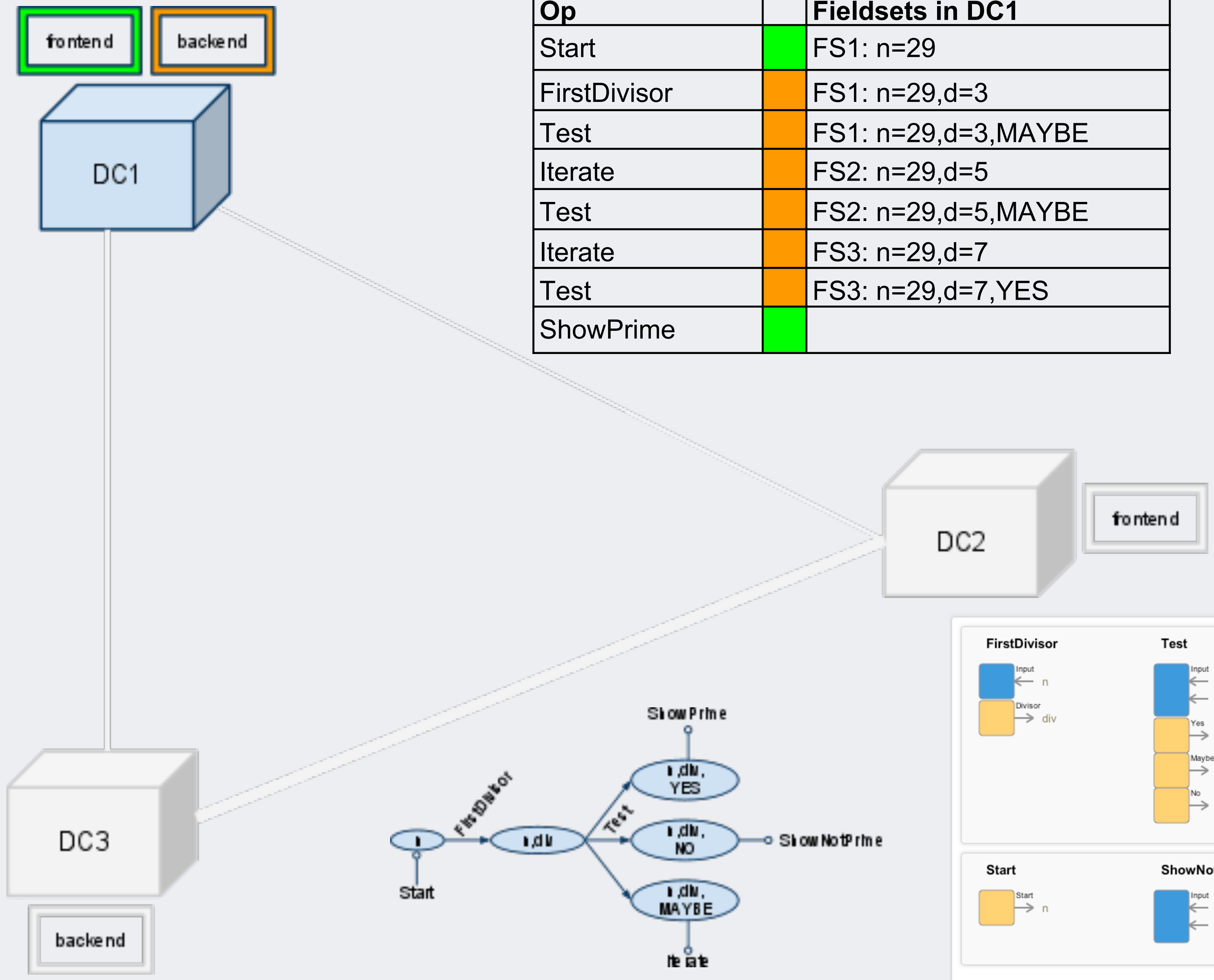
Example #3

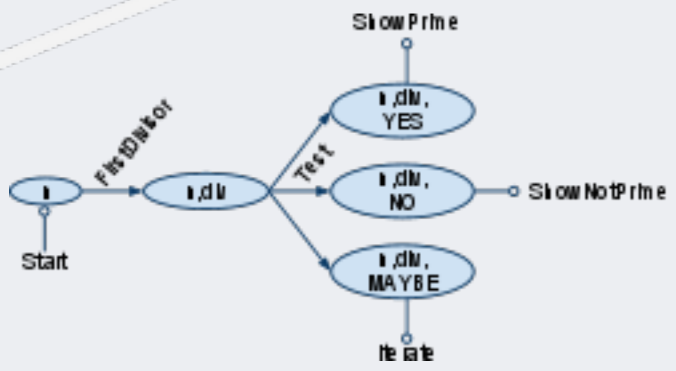

FirstDivisor
Input
n
Divisor
div
Test
Input
div
n
Yes
YES
Maybe
MAYBE
No
NO
Iterate
Input
div
MAYBE
n
Iterate
∞
div
n
backend
Start
Start
n
ShowNotPrime
Input
n
NO
ShowPrime
Input
n
YES
frontend

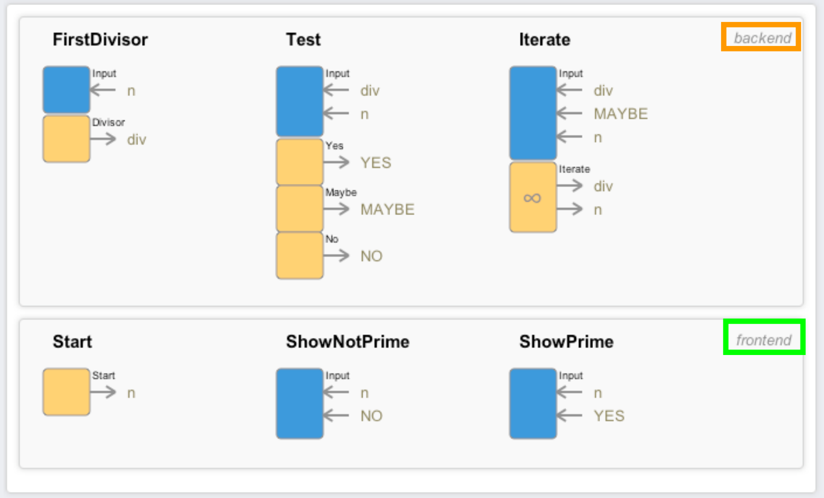

frontend
backend
DC1
ShowPrime
n,div,
YES
FirstDivisor
Test
n
n,div
n,div,
NO
ShowNotPrime
Start
n,div,
MAYBE
Iterate
DC2
frontend
DC3
backend

2 systems in 1 DC

Op		Fieldsets in DC1
Start	🟩	FS1: n=29
FirstDivisor	🟧	FS1: n=29,d=3
Test	🟧	FS1: n=29,d=3,MAYBE
Iterate	🟧	FS2: n=29,d=5
Test	🟧	FS2: n=29,d=5,MAYBE
Iterate	🟧	FS3: n=29,d=7
Test	🟧	FS3: n=29,d=7,YES
ShowPrime	🟩	



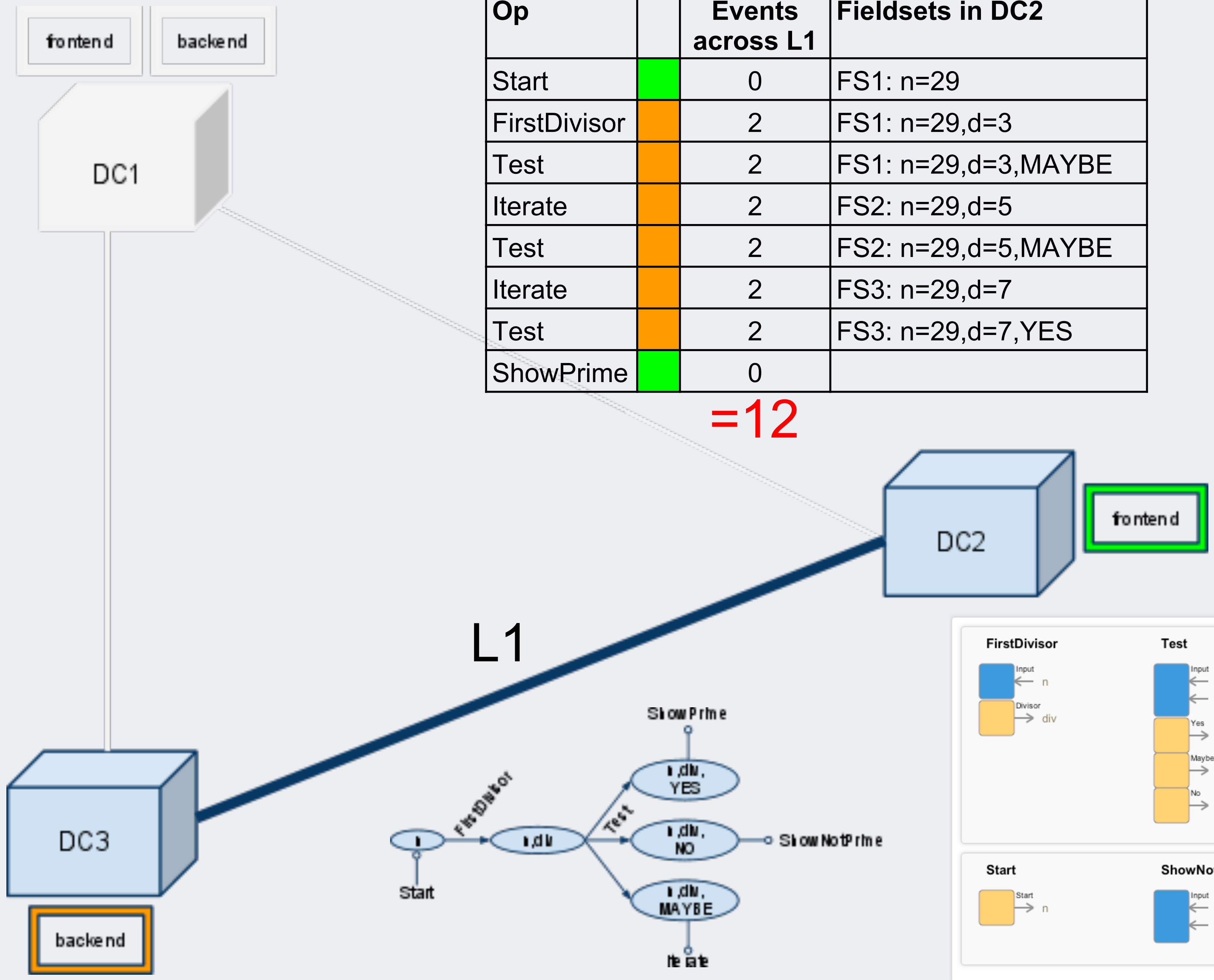


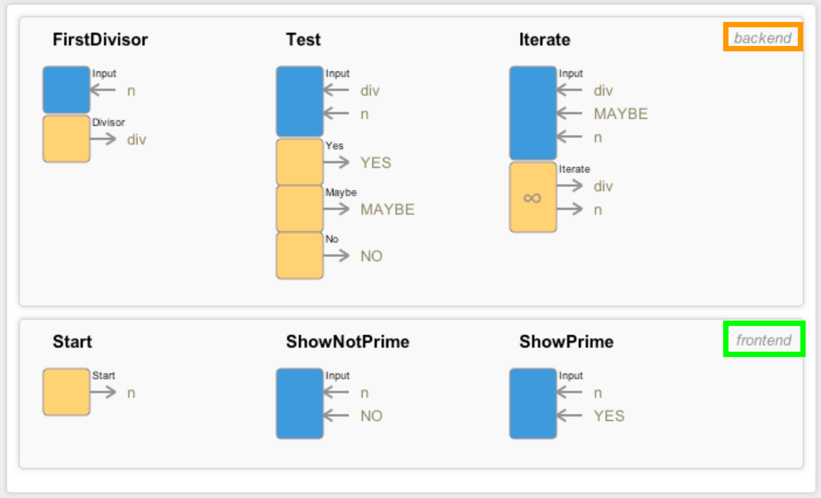


2 systems, 2 DCs, no distributed state

Op		Events across L1	Fieldsets in DC2
Start	🟩	0	FS1: n=29
FirstDivisor	🟧	2	FS1: n=29,d=3
Test	🟧	2	FS1: n=29,d=3,MAYBE
Iterate	🟧	2	FS2: n=29,d=5
Test	🟧	2	FS2: n=29,d=5,MAYBE
Iterate	🟧	2	FS3: n=29,d=7
Test	🟧	2	FS3: n=29,d=7,YES
ShowPrime	🟩	0	

=12

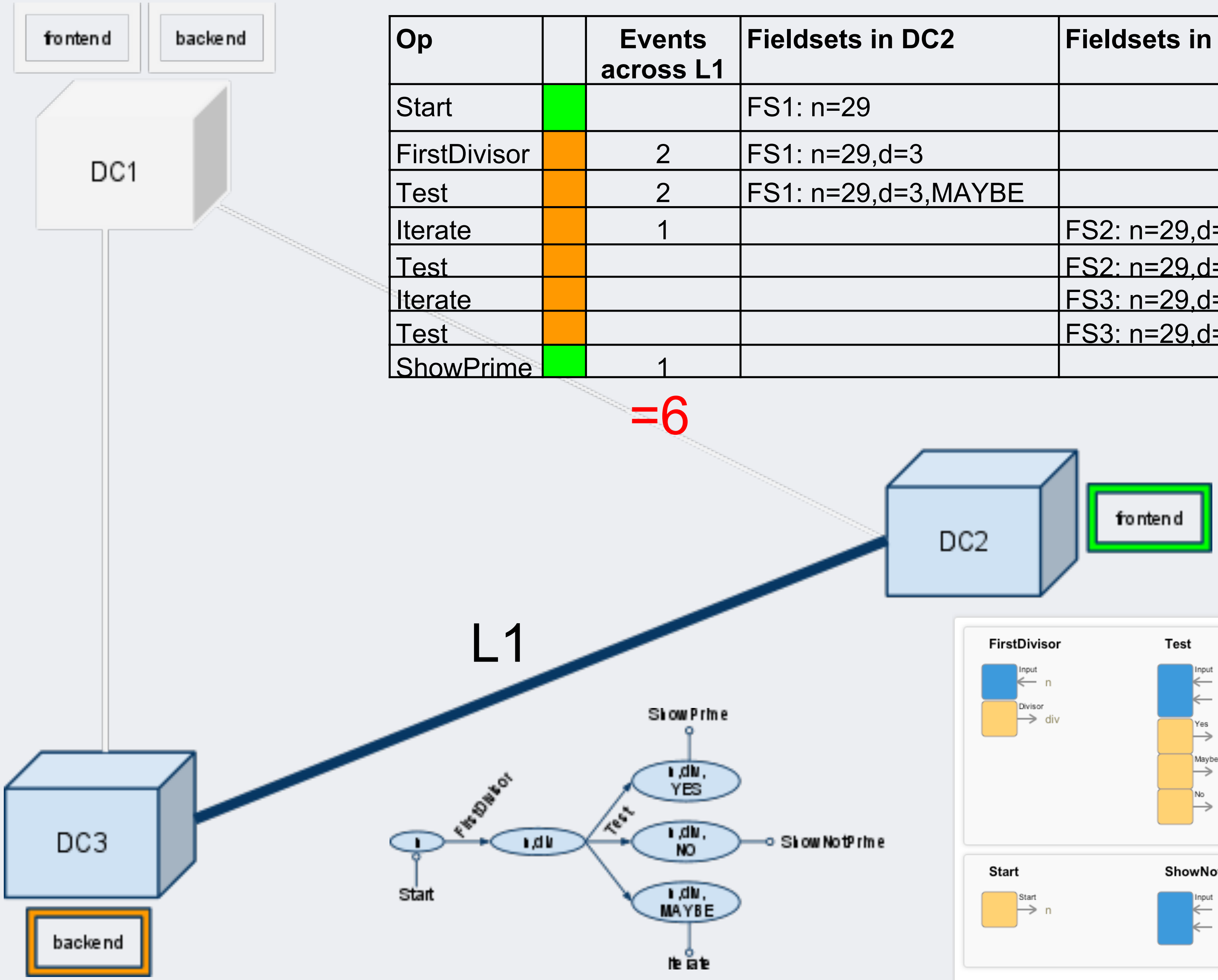


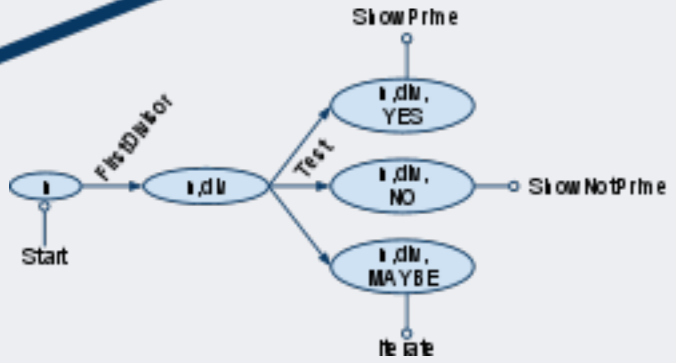


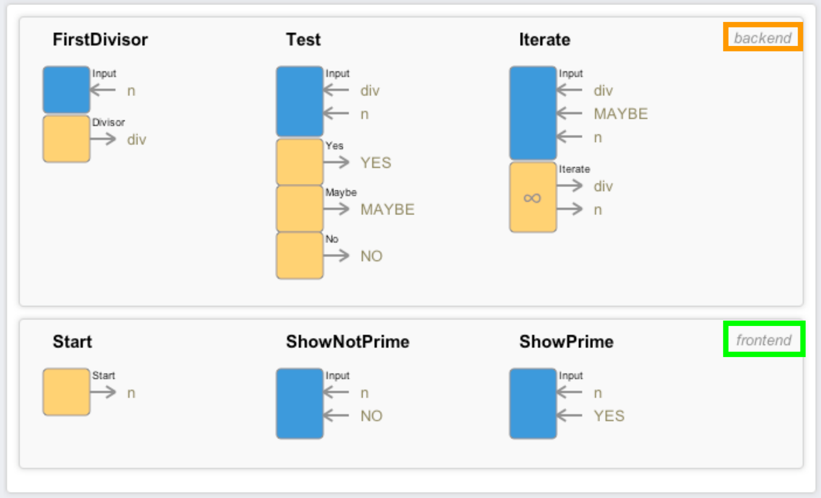
2 systems, 2 DCs, distributed state

Op		Events across L1	Fieldsets in DC2	Fieldsets in DC3
Start	🟩		FS1: n=29	
FirstDivisor	🟧	2	FS1: n=29,d=3	
Test	🟧	2	FS1: n=29,d=3,MAYBE	
Iterate	🟧	1		FS2: n=29,d=5
Test	🟧			FS2: n=29,d=5,MAYBE
Iterate	🟧			FS3: n=29,d=7
Test	🟧			FS3: n=29,d=7,YES
ShowPrime	🟩	1		

=6







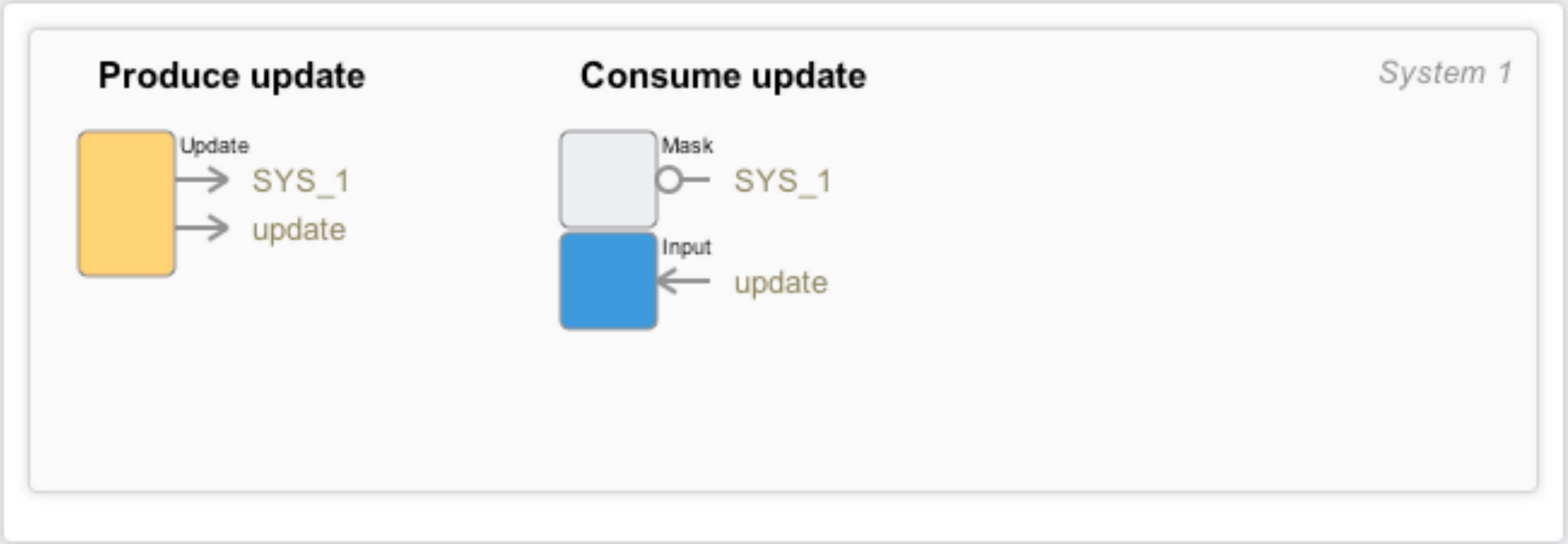
Pause
before
Masks, Mutability and Sync

Simple Sync

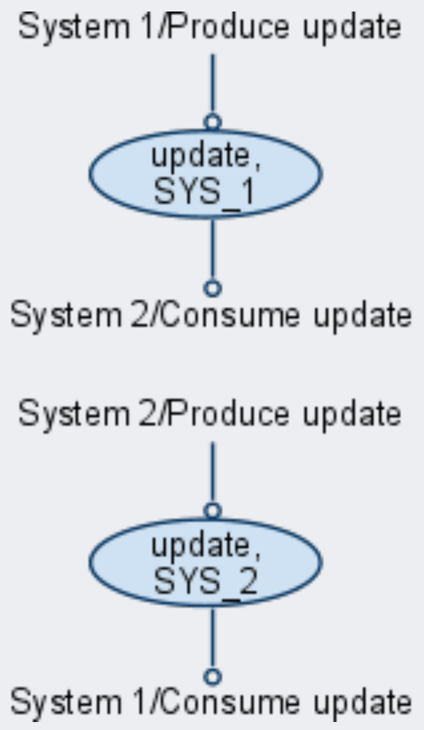
Here's a template for each system

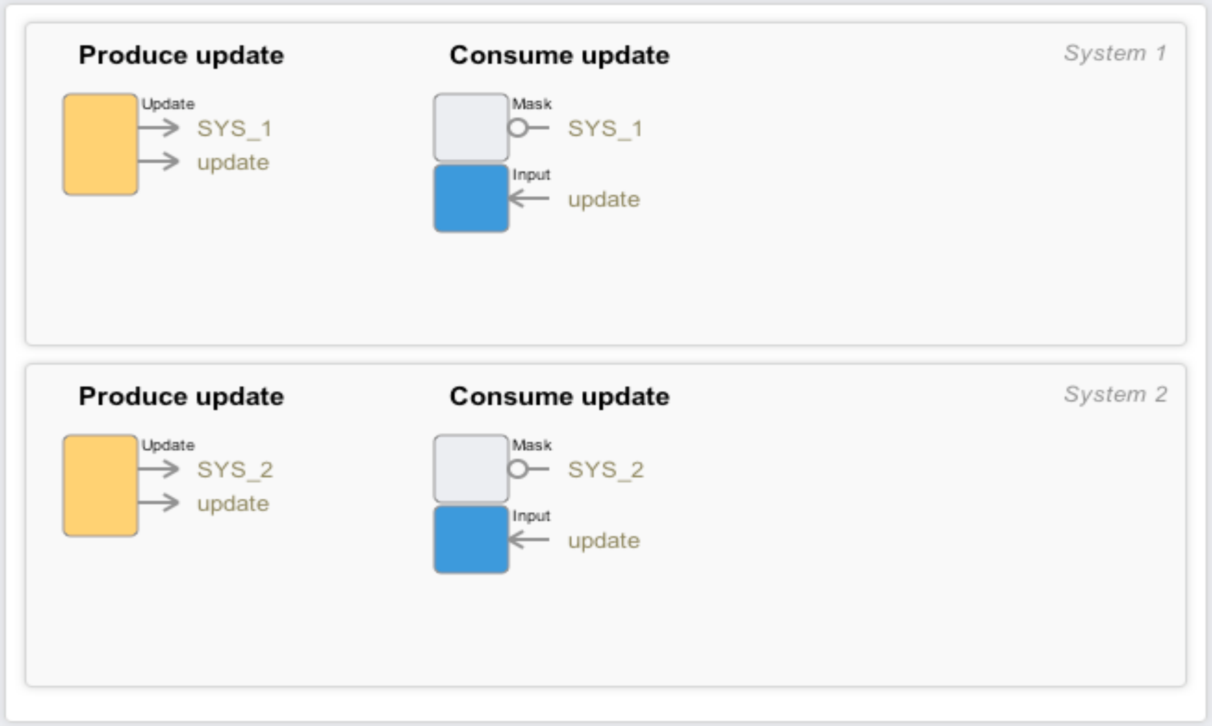
- update field is defined in parent folder
- SYS field is defined in template folder, ie unique per instance
    - Visibly disambiguated here by calling it SYS_1

Mask set means, this op can't happen if the masked fields are present on the fieldset.



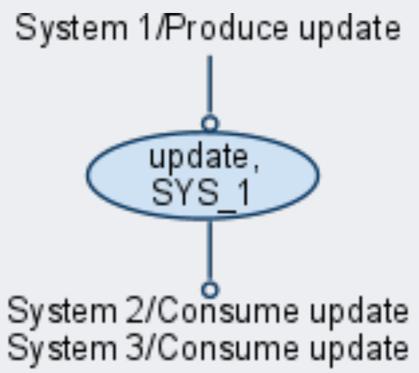
Simple Sync, 2 systems

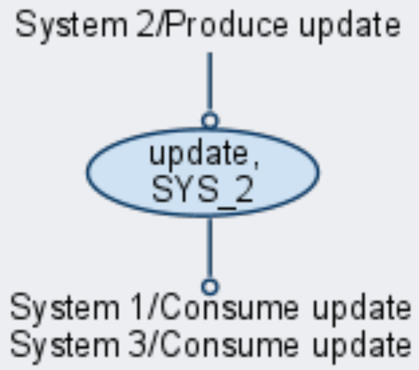


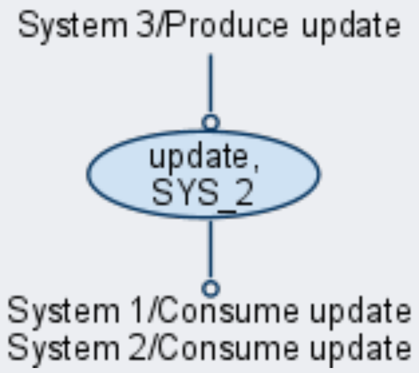


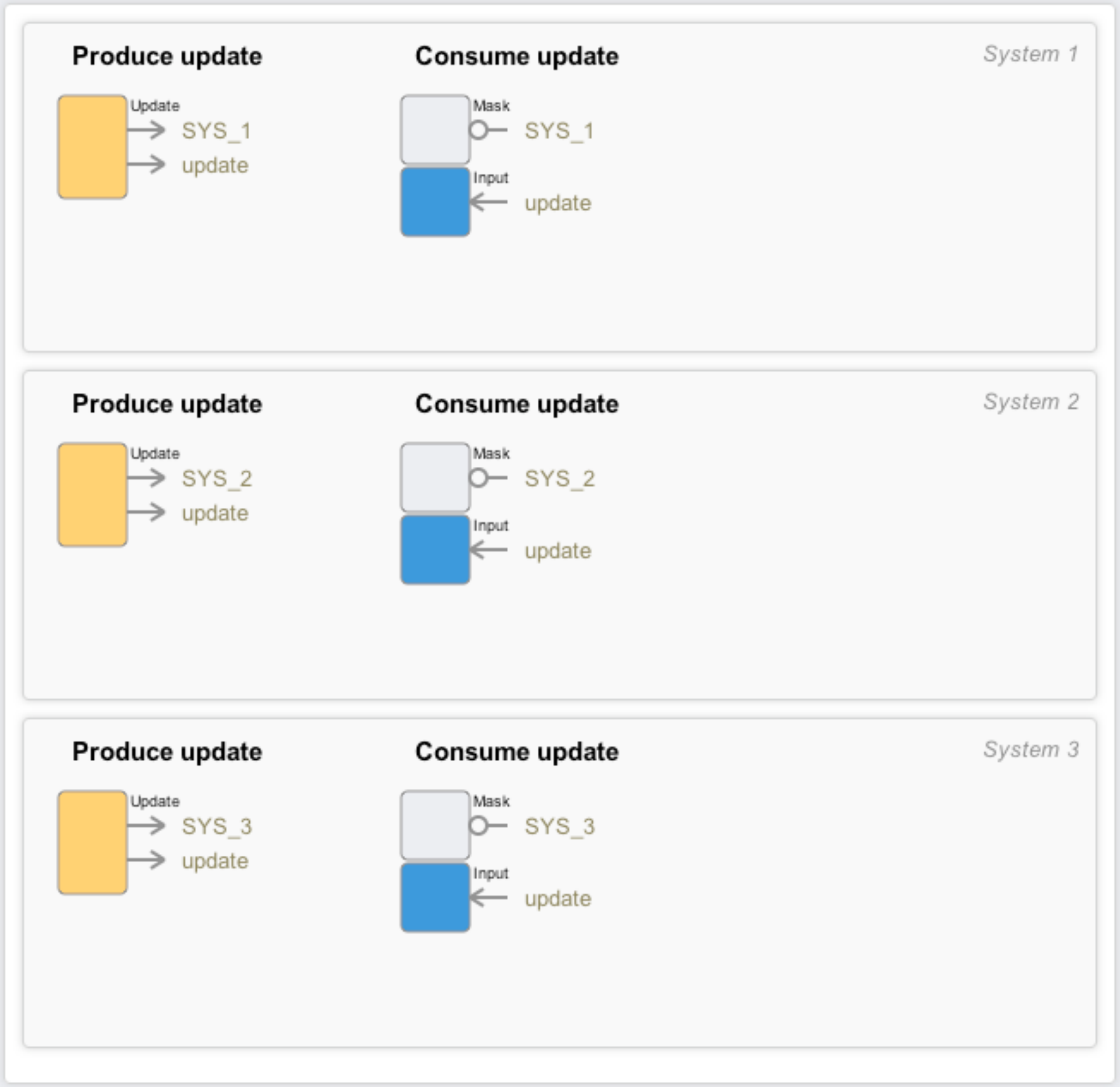
Mutable process - just drop in more systems...

Simple Sync, 3 systems



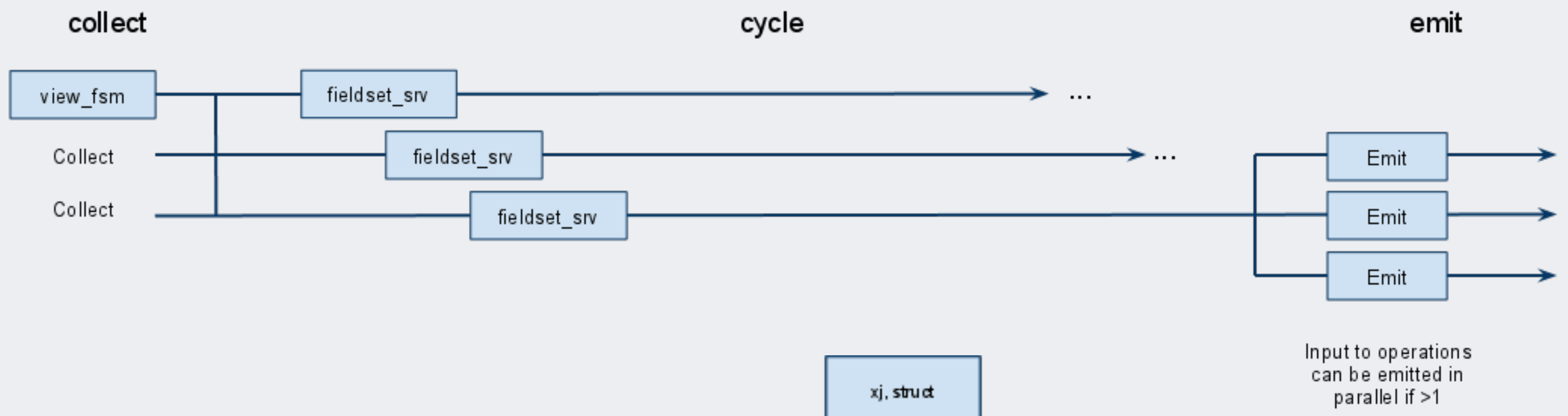


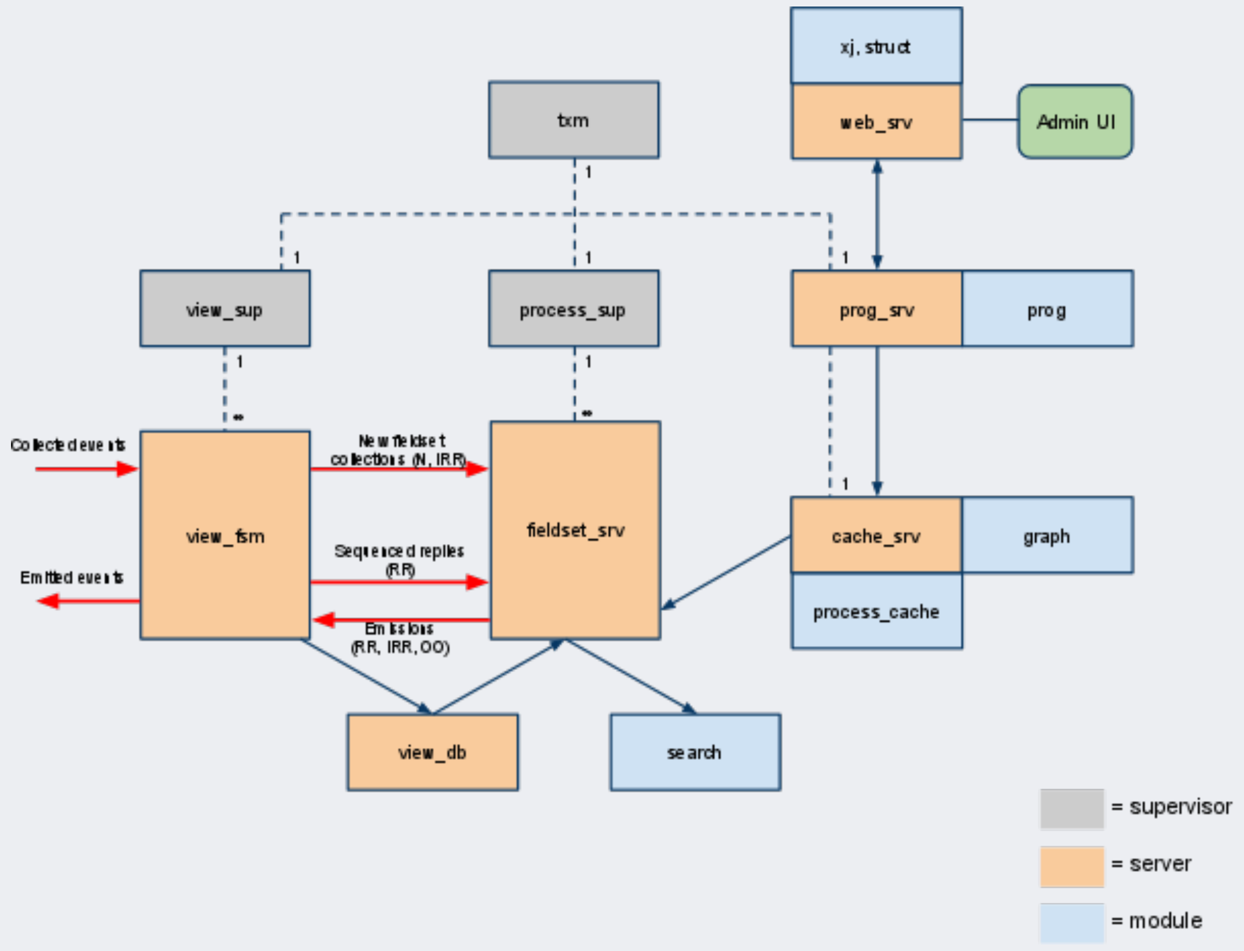




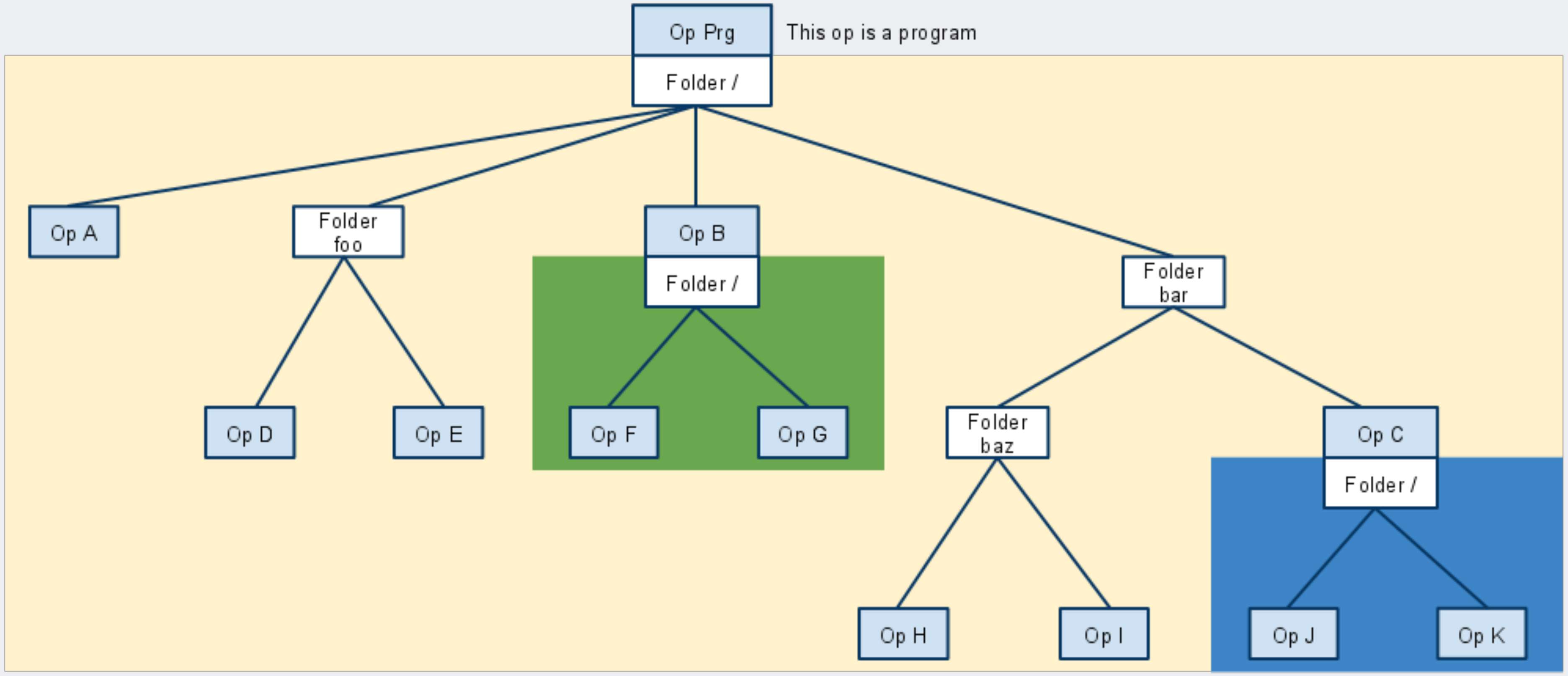
Pause
before
Erlang Implementation

Erlang Implementation





Mnesia Program Tree



🟨 Program prg

🟩 Program prg/B implements op /B

🟦 Program prg/bar/C implements op /bar/C

What follows?

Interesting use cases in Google

- Highly parallelized, interactive processes eg in ads
    - Complements batch mapreduces
- Call-out infrastructure for b2b2c applications

Mutable apps

- Set of ops is unordered
    - Add/remove ops at will for a mutable process
        - Synchronization applications

Apps built from search

- Set of ops is unordered
- Set of ops can have massive redundancy
    - Doesn't matter if there are only 10 useful ops out of 1000 in the unordered set
- Ops can self-render as (eg) web UI widgets
- So we *could* use search to build apps