Agenda

1. What is PSM?

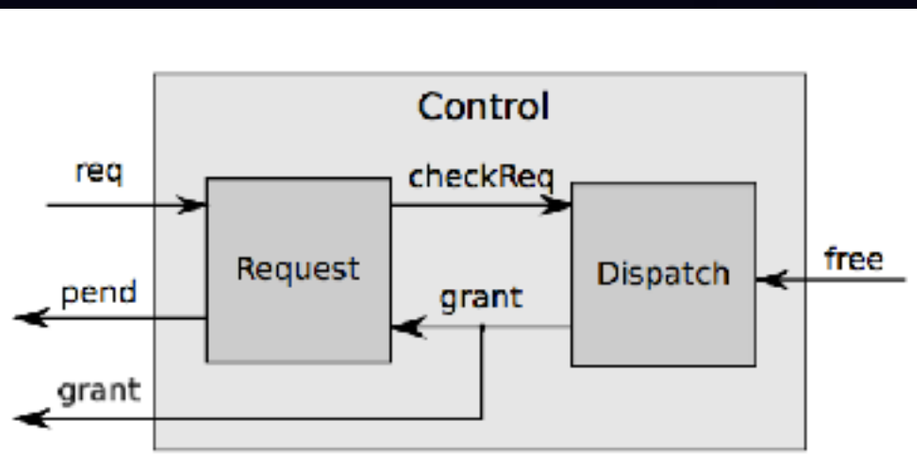
2. How to use the PSM Library

    - language combinators

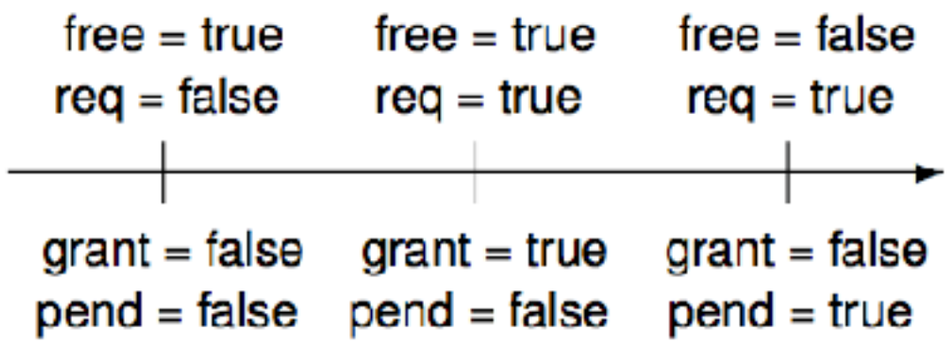
    - defining a PVar

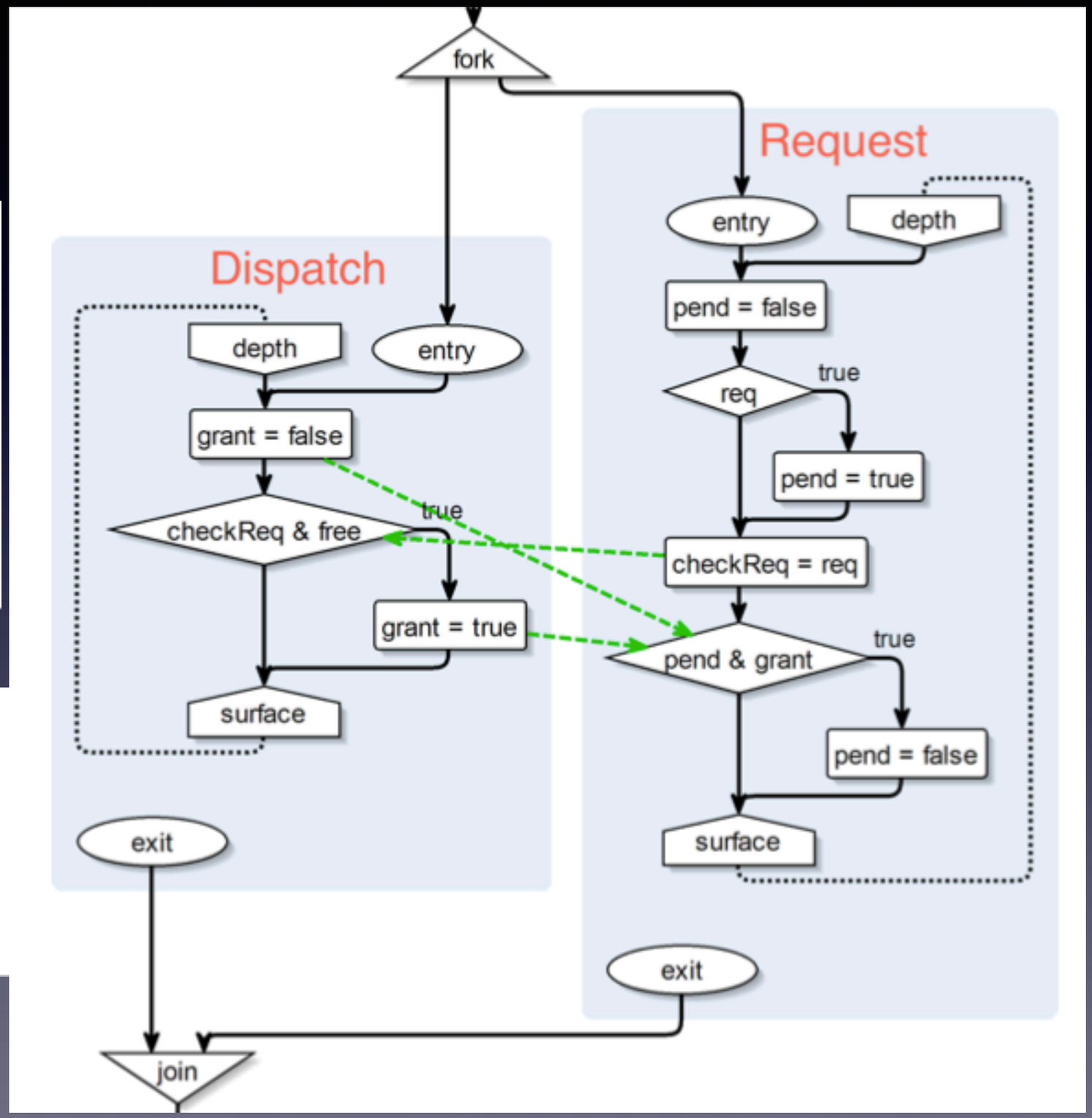
3. Demo

4. Conclusion



(c) The data-flow view





```
input: req = True
       free = True
```

Thread Dispatch

```
grant = False;

if (checkReq && free)

   grant = True;

pause;
```

Thread Request

```
pend = False;

if (req)

   pend = True;

checkReq = req;

if (pend && grant)

   pend = False;

pause;
```

```
output: grant = True
        pend = False
```

from: S. Smyth, C. Motika, K. Rathlev, R. von Hanxleden, and M. Mendler. 2017.
SCEst: Sequentially Constructive Esterel *ACM Trans. Embedd. Comput. Syst.* 1, 1, Article 1 (Jan 2017)

```
input: req = True
       free = True
```

Thread Dispatch

```
grant = False;

if (checkReq && free)

   grant = True;

pause;
```

Thread Request

```
pend = False;

if (req)

   pend = True;

checkReq = req;

if (pend && grant)

   pend = False;

pause;
```

```
output: grant = False
        pend = True
```

from: S. Smyth, C. Motika, K. Rathlev, R. von Hanxleden, and M. Mendler. 2017.
SCEst: Sequentially Constructive Esterel *ACM Trans. Embedd. Comput. Syst.* 1, 1, Article 1 (Jan 2017)

```
input: req = True
       free = True
```

Thread Dispatch

```
grant = False;

if (checkReq && free)

   grant = True;

pause;
```

Thread Request

```
pend = False;

if (req)

   pend = True;

checkReq = req;

if (pend && grant)

   pend = False;

pause;
```

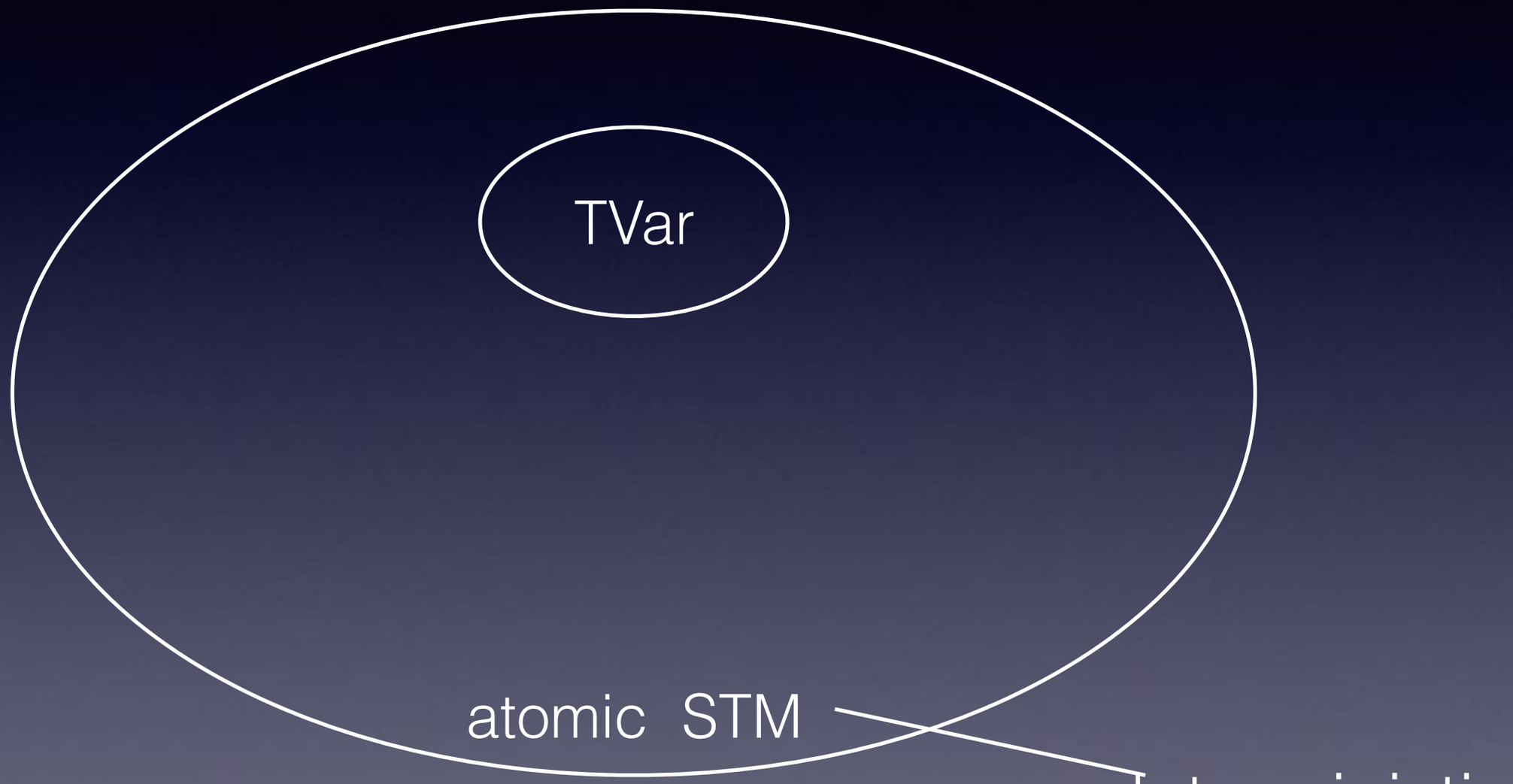
```
output: grant = True
        pend = True
```

1. What is PSM?

Policy Synchronised Memory

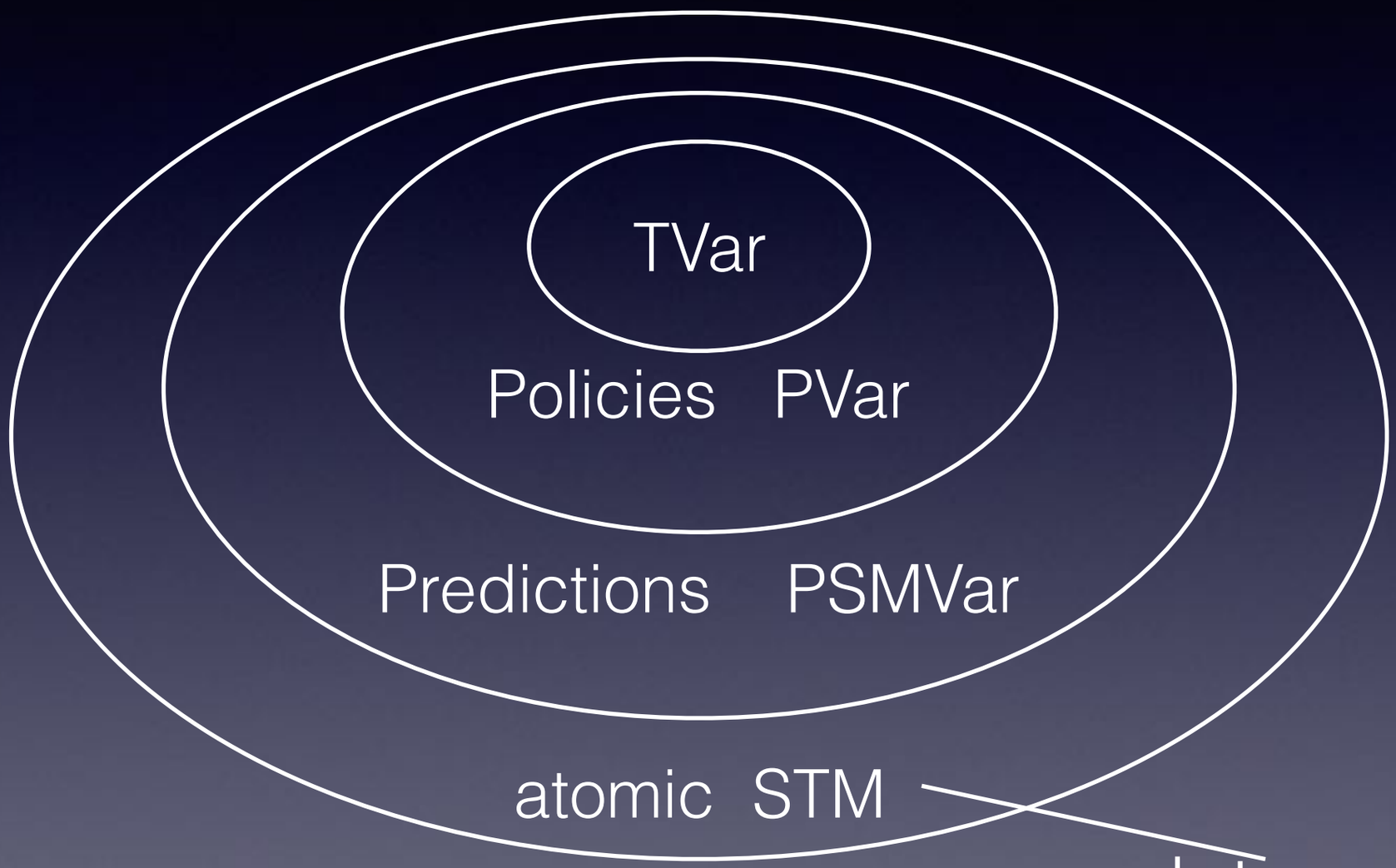
1. What is PSM?

Policy Synchronised Memory



1. What is PSM?

Policy Synchronised Memory



1. What is PSM?

type classes PVar and PVarMtd

$$\sum \Vdash \downarrow \mathrm{x.read}$$

- `admissible` depends on status

1. What is PSM?

type classes PVar and PVarMtd

$$\sum \Vdash \downarrow x.read$$

- `admissible` depends on status

$$\sum \Vdash x.write \dashrightarrow x.read$$

- methods may have `precedence` over each other

1. What is PSM?

type classes PVar and PVarMtd

$$\sum \Vdash \downarrow x.read$$

- `admissible` depends on status

$$\sum \Vdash x.write \dashrightarrow x.read$$

- methods may have `precedence` over each other

$$\sum; E \Vdash \downarrow x.read$$

- `enabled`: admissible AND unblocked under arbitrary actions of environment E

1. What is PSM?

PSMVar

PSMVars store state and prediction information

MtdId \ Thread	Thread1	Thread2
1 (write)	0	1
2 (read)	2	0

2. How to use PSM

implementation in Haskell

- language combinators
- defining a PVar

2. How to use PSM

language combinators

- `newPVar :: PVar a => PVarStat a -> (PSMVar a -> PSM b) -> PSM b`
  instantiates a new local PVar
- `nothing :: PSM ()`
  empty PSM process
- `pause :: PSM ()`
  pauses thread; wait for tick
- `(|||) :: PSM () -> PSM () -> PSM ()`
  parallel composition (concurrent threads)
- `(>>>) :: PSM a -> PSM b -> PSM b`
  sequential composition, ignoring value
- `(>>>=) :: PSM a -> (PSMVal a -> PSM b) -> PSM b`
  sequential composition with binding of return value
- `ifte :: Bool -> PSM a -> PSM a -> PSM a`
  conditional branching - if then else

2. How to use PSM

defining a PVar

```
0   -- SCBoolOR
1   data SCBool = SCBool (TVar Bool)
2
3   type PSCBool = PSMVar SCBool
```

MtdId

```
39  setPSCBool :: PSMVar SCBool -> PSMVal Bool -> PSM ()
53  updPSCBool :: PSMVar SCBool -> PSMVal Bool -> PSM ()
67  readPSCBool :: PSMVar SCBool -> PSM Bool
```

0
1
2

2. How to use PSM

defining a PVar

```
prec :: SCBool -> MtdId -> STM (Maybe [MtdId])
```

```haskell
07  instance PSMType SCBool where
08
09  -- setSCBool MtdId 0
10    prec _ (MtdId 0) = return $ Just [MtdId 0]
11  -- updSCBool MtdId 1
12    prec _ (MtdId 1) = return $ Just [MtdId 0]
13  -- readSCBool MtdId 2
14    prec _ (MtdId 2) = return $ Just [MtdId 0, MtdId 1]
```

2. How to use PSM

defining a PVar

```
18  instance PVar SCBool where
19    type PVarStat SCBool = ()
21    init _ = do
22      mustRef <- atomically $ newTVar False
23      return $ SCBool mustRef
25    tick _ = return ()   -- reset??
27    term _ = return ()   -- clean-up??
```

```
init :: PVarStat a -> IO a

tick :: a -> IO ()

term :: a -> IO ()
```

2. How to use PSM

```haskell
39  setPSCBool :: PSMVar SCBool -> PSMVal Bool -> PSM ()
40  setPSCBool scbool v = method SetSCBool scbool v
```

```haskell
32  instance PVarMtd SCBool SetSCBool where
33    type PVarMtdIn SCBool SetSCBool  = Bool
34    type PVarMtdOut SCBool SetSCBool = ()
35    method _ = runSTMMethod (MtdId 0) SetSCBool _must where
36      _must _ (SCBool scboolRef) v =
37        writeTVar scboolRef v
```

```haskell
method :: m -> PSMVar p -> PSMVal (PVarMtdIn p m) ->
          PSM (PVarMtdOut p m)

type PVarMtdIn p m :: *

type PVarMtdOut p m :: *
```

2. How to use PSM

```haskell
53  updPSCBool :: PSMVar SCBool -> PSMVal Bool -> PSM ()
54  updPSCBool scbool v = method UpdSCBool scbool v
```

```haskell
45  instance PVarMtd SCBool UpdSCBool where
46    type PVarMtdIn SCBool UpdSCBool = Bool
47    type PVarMtdOut SCBool UpdSCBool = ()
48    method _ = runSTMMethod (MtdId 1) UpdSCBool _must where
49      _must _ (SCBool scboolRef) v = do
50        scbool <- readTVar scboolRef
51        writeTVar scboolRef ((||) scbool v)
```

```haskell
method :: m -> PSMVar p -> PSMVal (PVarMtdIn p m) ->
          PSM (PVarMtdOut p m)

type PVarMtdIn p m :: *

type PVarMtdOut p m :: *
```

2. How to use PSM

```haskell
67  readPSCBool :: PSMVar SCBool -> PSM Bool
68  readPSCBool scbool = method ReadSCBool scbool (PSMVal ())
```

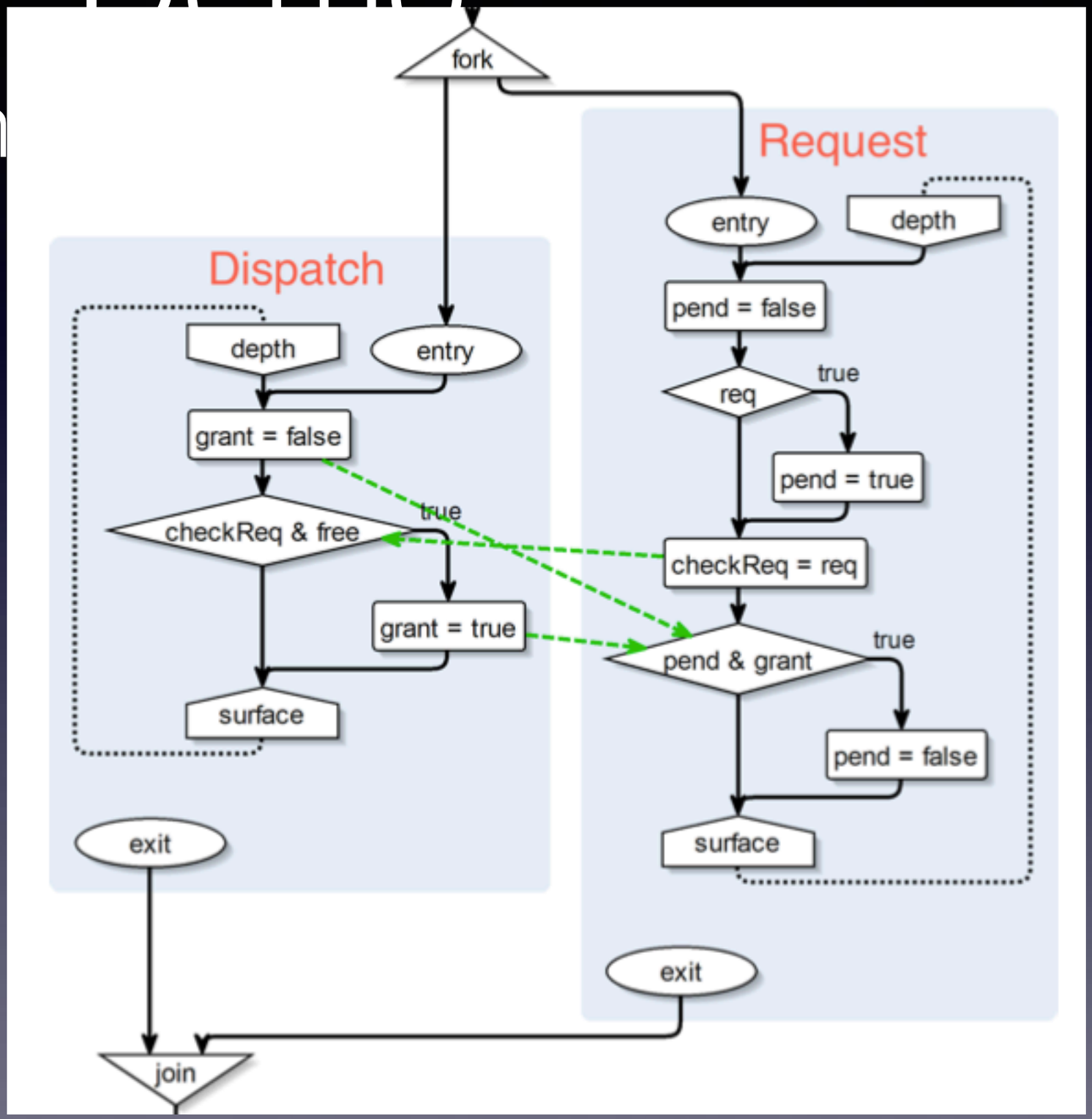
```haskell
59  instance PVarMtd SCBool ReadSCBool where
60    type PVarMtdIn SCBool ReadSCBool = ()
61    type PVarMtdOut SCBool ReadSCBool = Bool
62    method _ = runSTMMethod (MtdId 2) ReadSCBool _must where
63      _must _ (SCBool scboolRef) _ = do
64        scbool <- readTVar scboolRef
65        return scbool
```

```haskell
method :: m -> PSMVar p -> PSMVal (PVarMtdIn p m) ->
          PSM (PVarMtdOut p m)

type PVarMtdIn p m :: *

type PVarMtdOut p m :: *
```

3. Demo
control example

3 Demo

con



Conclusion

- currently single clock
  -> multi-rate clocks for each thread planned
- pause σ
- [P]σ clock ignore
- miniSCoL compiler

Thank you for your attention

```
*Test10> knockKnock
>>
Tick 0:
        Knock, knock!
        Race condition.
        Who's there?
>>
Tick 1:
        Knock, knock!
        Who's there?
        Race condition.
>>
```