Block on🤘

(WIP) Ideas for primitive blocking: from locks to futexes and beyond

Justus Fasse and Bart Jacobs

June 5

Agenda

Problem: classic deadlock-freedom specs are too restrictive

Solution: Parameterize modules by client-specific deadlock-freedom argument

Application: Futexes (Compare-and-sleep)

Outlook

Agenda

Problem: classic deadlock-freedom specs are too restrictive

Solution: Parameterize modules by client-specific deadlock-freedom argument

Application: Futexes (Compare-and-sleep)

Outlook

Primitive blocking

Operational semantics (stuttering)

Acq-Succ

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{false}}{(\texttt{Acq}\ lk, \sigma) \to_{\mathsf{h}} ((), \sigma : \textsc{Heap}[lk \leftarrow \texttt{true}], \epsilon)}$$

Primitive blocking

Operational semantics (stuttering)

Acq-Succ

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{false}}{(\texttt{Acq}\ lk, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \textsc{Heap}[lk \leftarrow \texttt{true}], \epsilon)}$$

Acq-Block

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{true}}{(\texttt{Acq}\ lk, \sigma) \rightarrow_{\mathsf{h}} (\texttt{Acq}\ lk, \sigma, \epsilon)}$$

Primitive blocking

Operational semantics (stuttering)

$$\textsc{Acq-Succ}\quad \frac{\sigma.\textsc{Heap}(lk) = \texttt{false}}{(\texttt{Acq}\ lk, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \textsc{Heap}[lk \leftarrow \texttt{true}], \epsilon)}$$

$$\textsc{Acq-Block}\quad \frac{\sigma.\textsc{Heap}(lk) = \texttt{true}}{(\texttt{Acq}\ lk, \sigma) \rightarrow_{\mathsf{h}} (\texttt{Acq}\ lk, \sigma, \epsilon)}$$

$$\textsc{Rel}\quad \frac{\sigma.\textsc{Heap}(lk) = \texttt{true}}{(\texttt{Rel}\ lk, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \textsc{Heap}[lk \leftarrow \texttt{false}], \epsilon)}$$

“Classic” obligations reasoning

Operational semantics

NewLock

$$\frac{lk \notin \mathrm{dom}(\sigma.\textsc{Locks})}{(\texttt{NewLock}\ \lambda, \sigma) \underset{\theta}{\rightarrow}_{\mathtt{h}} (lk, \sigma : \textsc{Locks}[lk \leftarrow (\texttt{false}, \lambda)], \epsilon)}$$

“Classic” obligations reasoning

Operational semantics

NewLock

$$\frac{lk \notin \mathrm{dom}(\sigma.\textsc{Locks})}{(\texttt{NewLock}\ \lambda, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} (lk, \sigma : \textsc{Locks}[lk \leftarrow (\texttt{false}, \lambda)], \epsilon)}$$

Acq-Succ

$$\frac{\sigma.\textsc{Locks}(lk) = (\texttt{false}, \lambda)}{(\texttt{Acq}\ lk, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} ((), \sigma : \textsc{Locks}[lk \leftarrow (\texttt{true}, \lambda)] : \textsc{Obs}(\theta) \uplus\!\!\hookleftarrow \{\lambda\}, \epsilon)}$$

“Classic” obligations reasoning

Operational semantics

NewLock

$$\frac{lk \notin \mathrm{dom}(\sigma.\textsc{Locks})}{(\texttt{NewLock}\ \lambda, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} (lk, \sigma : \textsc{Locks}[lk \leftarrow (\texttt{false}, \lambda)], \epsilon)}$$

Acq-Succ

$$\frac{\sigma.\textsc{Locks}(lk) = (\texttt{false}, \lambda)}{(\texttt{Acq}\ lk, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} ((), \sigma : \textsc{Locks}[lk \leftarrow (\texttt{true}, \lambda)] : \textsc{Obs}(\theta) \uplus\!\!\hookleftarrow \{\lambda\}, \epsilon)}$$

Rel

$$\frac{\sigma.\textsc{Locks}(lk) = (\texttt{true}, \lambda)}{(\texttt{Rel}\ lk, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} ((), \sigma : \textsc{Locks}[lk \leftarrow (\texttt{false}, \lambda)] : \textsc{Obs}(\theta) \setminus\!\!\hookleftarrow \{\lambda\}, \epsilon)}$$

“Classic” obligations reasoning

Operational semantics

NewLock

$$\frac{lk \notin \mathrm{dom}(\sigma.\textsc{Locks})}{(\texttt{NewLock}\ \lambda, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} (lk, \sigma : \textsc{Locks}[lk \leftarrow (\texttt{false}, \lambda)], \epsilon)}$$

Acq-Succ

$$\frac{\sigma.\textsc{Locks}(lk) = (\texttt{false}, \lambda)}{(\texttt{Acq}\ lk, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} ((), \sigma : \textsc{Locks}[lk \leftarrow (\texttt{true}, \lambda)] : \textsc{Obs}(\theta) \uplus\!\hookleftarrow \{\lambda\}, \epsilon)}$$

Acq-Block

$$\frac{\sigma.\textsc{Heap}(lk) = (\texttt{true}, \lambda) \qquad \lambda \prec \sigma.\textsc{Obs}(\theta)}{(\texttt{Acq}\ lk, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} (\texttt{Acq}\ lk, \sigma, \epsilon)}$$

Rel

$$\frac{\sigma.\textsc{Locks}(lk) = (\texttt{true}, \lambda)}{(\texttt{Rel}\ lk, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} ((), \sigma : \textsc{Locks}[lk \leftarrow (\texttt{false}, \lambda)] : \textsc{Obs}(\theta) \setminus\!\hookleftarrow \{\lambda\}, \epsilon)}$$

“Classic” obligations reasoning

Operational semantics

NewLock

$$\frac{lk \notin \mathrm{dom}(\sigma.\textsc{Locks})}{(\texttt{NewLock}\ \lambda, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} (lk, \sigma : \textsc{Locks}[lk \leftarrow (\texttt{false}, \lambda)], \epsilon)}$$

Acq-Succ

$$\frac{\sigma.\textsc{Locks}(lk) = (\texttt{false}, \lambda)}{(\texttt{Acq}\ lk, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} ((), \sigma : \textsc{Locks}[lk \leftarrow (\texttt{true}, \lambda)] : \textsc{Obs}(\theta) \uplus\hookleftarrow \{\lambda\}, \epsilon)}$$

Acq-Block

$$\frac{\sigma.\textsc{Heap}(lk) = (\texttt{true}, \lambda) \qquad \boxed{\lambda \prec \sigma.\textsc{Obs}(\theta)}}{(\texttt{Acq}\ lk, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} (\texttt{Acq}\ lk, \sigma, \epsilon)}$$

Rel

$$\frac{\sigma.\textsc{Locks}(lk) = (\texttt{true}, \lambda)}{(\texttt{Rel}\ lk, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} ((), \sigma : \textsc{Locks}[lk \leftarrow (\texttt{false}, \lambda)] : \textsc{Obs}(\theta) \setminus\hookleftarrow \{\lambda\}, \epsilon)}$$

“Classic” obligations reasoning
Operational semantics

Fork

$$\frac{\theta' \notin \mathrm{dom}(\Theta)}{(\texttt{fork}\ e\ obs, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} ((), \sigma : \textsc{Obs}(\theta) \setminus\!\hookleftarrow obs : \textsc{Obs}(\theta') \uplus\!\hookleftarrow obs, (\theta', (e; \texttt{Finish})))}$$

Finish

$$\frac{\sigma.\textsc{Obs}(\theta) = \emptyset}{(\texttt{Finish}, \sigma) \underset{\theta}{\rightarrow}_{\mathsf{h}} ((), \sigma, \epsilon)}$$

À la Kobayashi 2006, Leino et al. 2010, Boström and Müller 2015, Jacobs et al. 2018, Reinhard and Jacobs 2021

Obligations

A deadlock-free language, restrictive language

```
// obs(∅)
Acq lk;
// obs({lk.lev})
Rel lk;
// obs(∅)
Finish
```

||

```
// obs(∅)
Acq lk;
// obs({lk.lev})
Rel lk;
// obs(∅)
Finish
```

Obligations

A deadlock-free language, restrictive language

// *obs(∅)*
Acq lk;
// *obs({lk.lev})*
Rel lk;
// *obs(∅)*
Finish

‖

// *obs(∅)*
Acq lk;
// *obs({lk.lev})*
Rel lk;
// *obs(∅)*
Finish

// *obs(∅)*
Acq lk;
// *obs({lk.lev})*
~~Rel(x);~~
Finish

‖

// *obs(∅)*
Acq lk;
// *obs({lk.lev})*
Rel lk;
// *obs(∅)*
Finish

Obligations

A deadlock-free language, restrictive language

```
// obs(∅)
Acq lk;
// obs({lk.lev})
Rel lk;
// obs(∅)
Finish
```

‖

```
// obs(∅)
Acq lk;
// obs({lk.lev})
Rel lk;
// obs(∅)
Finish
```

// obs(∅)
Acq lk;
// obs({lk.lev})
~~Rel(x);~~
Finish

‖

```
// obs(∅)
Acq lk;
// obs({lk.lev})
Rel lk;
// obs(∅)
Finish
```

Unfulfilled obligation

Obligations

A deadlock-free language, restrictive language

```
// obs(∅)
Acq lk;
// obs({lk.lev})
Rel lk;
// obs(∅)
Finish
```

‖

```
// obs(∅)
Acq lk;
// obs({lk.lev})
Rel lk;
// obs(∅)
Finish
```

```
// obs({x.lev})
Acq y;
Rel x;
Finish
```

‖

```
// obs({y.lev})
Acq x;
Rel y;
Finish
```

// obs(∅)
Acq lk;
// obs({lk.lev})
~~Rel(x);~~
Finish

‖

```
// obs(∅)
Acq lk;
// obs({lk.lev})
Rel lk;
// obs(∅)
Finish
```

Unfulfilled obligation

Obligations

A deadlock-free language, restrictive language

```
// obs(∅)              ||  // obs(∅)
Acq lk;                ||  Acq lk;
// obs({lk.lev})       ||  // obs({lk.lev})
Rel lk;                ||  Rel lk;
// obs(∅)              ||  // obs(∅)
Finish                 ||  Finish
```

```
// obs({x.lev})  ||  // obs({y.lev})
Acq y;           ||  Acq x;
Rel x;           ||  Rel y;
Finish           ||  Finish
```

Failing level check

// obs(∅)
Acq lk;
// obs({lk.lev})
~~Rel(x);~~
Finish

\|\|

```
// obs(∅)
Acq lk;
// obs({lk.lev})
Rel lk;
// obs(∅)
Finish
```

Unfulfilled obligation

Classic lock specifications

NewLockClassic

$\{R\}$ **NewLock** λ $\{lk.\ \text{is\_lock}(lk, \lambda, R)\}$

AcqClassic

$\{\text{is\_lock}(lk, \lambda, R) * \text{obs}(O) * \lambda \prec \text{obs}(O)\}$ **Acq** lk $\{\text{obs}(O \uplus \{\lambda\}) * \text{locked}(lk, \lambda, R) * R\}$

RelClassic

$\{\text{obs}(O) * \text{locked}(lk, \lambda, R) * R\}$ **Rel** lk $\{\text{obs}(O \setminus \{\lambda\})\}$

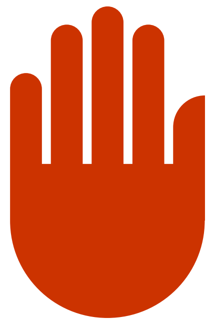
Finish

$\{\text{obs}(\emptyset)\}$ **Finish** $\{\text{True}\}$

External knowledge of deadlock-freedom

```
// obs(∅)
Acq x;
// obs({lk.lev})
Finish;
```

External knowledge of deadlock-freedom



```
// obs(∅)
Acq x;
// obs({lk.lev})
Finish;
```

Unfulfilled obligation

Lock handoffs

`exit 0` kills the program in a “good” state

```
let x = NewLock () in
let f = ref true in

      // obs({∅})
      if !f then exit 0;
      // obs({∅})
      Rel x;
      Finish;
```

Lock handoffs

`exit 0` kills the program in a “good” state

```
let x = NewLock () in
let f = ref true in
```

```
// obs({∅})            ||  // obs({∅})
Acq x;                 ||  if !f then exit 0;
// obs({lk.lev})       ||  // obs({∅})
f := false;            ||  Rel x;
Finish;                ||  Finish;
```

Lock handoffs

`exit 0` kills the program in a “good” state

```
let x = NewLock () in
let f = ref true in
```

```
// obs({∅})                 ||  // obs({∅})
Acq x;                      ||  if !f then exit 0;
// obs({lk.lev})            ||  // obs({∅})
f := false;                 ||  Rel x;
Finish;                     ||  Finish;
```

Lock handoffs

`exit 0` kills the program in a “good” state

```
let x = NewLock () in
let f = ref true in
```

```
// obs({∅})           ||  // obs({∅})
Acq x;                ||  if !f then exit 0;
// obs({lk.lev})      ||  // obs({∅})
f := false;           ||  Rel x;
Finish;               ||  Finish;
```

Lock handoffs

`exit 0` kills the program in a "good" state

```
let x = NewLock () in
let f = ref true in
```

```
// obs({∅})
Acq x;
// obs({lk.lev})
f := false;
Finish;
```

∥

```
// obs({∅})
if !f then exit 0;
// obs({∅})
Rel x;
Finish;
```

∥

```
// obs({∅})
Acq x;
// obs({lk.lev})
Rel x;
// obs({∅})
Finish;
```

Lock handoffs

`exit 0` kills the program in a “good” state

```
let x = NewLock () in
let f = ref true in
```

```
// obs({∅})
Acq x;
// obs({lk.lev})
f := false;
Finish;
```

||

```
// obs({∅})
if !f then exit 0;
// obs({∅})
Rel x;
Finish;
```

||

```
// obs({∅})
Acq x;
// obs({lk.lev})
Rel x;
// obs({∅})
Finish;
```

Unfulfilled obligation

Lock handoffs

`exit 0` kills the program in a “good” state

```
let x = NewLoc…
let f = r…

// obs({∅})
Acq x…
…
…ish;

…obs({∅})
Acq x;
// obs({lk.lev})
Rel x;
// obs({∅})
Finish;
```

Too restrictive

Unfulfilled obligation

Deadlocked?

```
let x = NewLock () in
let f = ref true in
```

```
Acq x;              || if !f then exit 0;  || Acq x;
// Critical section || // Critical section || // Critical section
f := false;         || Rel x;              || Rel x;
Finish;             || Finish;             || Finish;
```

Deadlock

Fork

$(\texttt{fork}\ e, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \#\textsc{Alive}{+}{+}, (e; \texttt{Finish}))$

Deadlock

Fork

$$(\texttt{fork}\ e, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \#\textsc{Alive}{+}{+}, (e; \texttt{Finish}))$$

Finish

$$\frac{\sigma.\#\textsc{Waiting} = 0 \lor \sigma.\#\textsc{Alive} - 1 > \sigma.\#\textsc{Waiting}}{(\texttt{Finish}, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \#\textsc{Alive}{-}{-}, \epsilon)}$$

Deadlock

Fork

$$(\texttt{fork}\ e, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \#\textsc{Alive}{+}{+}, (e; \texttt{Finish}))$$

Acq-Succ

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{false}}{(\texttt{Acq}\ lk, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \textsc{Heap}[lk \leftarrow \texttt{true}], \epsilon)}$$

Finish

$$\frac{\sigma.\#\textsc{Waiting} = 0 \lor \sigma.\#\textsc{Alive} - 1 > \sigma.\#\textsc{Waiting}}{(\texttt{Finish}, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \#\textsc{Alive}{-}{-}, \epsilon)}$$

Deadlock

Fork

$$(\texttt{fork}\ e, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \#\textsc{Alive}{+}{+}, (e; \texttt{Finish}))$$

Acq-Succ

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{false}}{(\texttt{Acq}\ lk, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \textsc{Heap}[lk \leftarrow \texttt{true}], \epsilon)}$$

Acq-Wait

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{true} \qquad \sigma.\#\textsc{Alive} > \sigma.\#\textsc{Waiting} + 1}{(\texttt{Acq}\ lk, \sigma) \rightarrow_{\mathsf{h}} (\texttt{WAIT}\ lk, \sigma : \#\textsc{Waiting}{+}{+}, \epsilon)}$$

Finish

$$\frac{\sigma.\#\textsc{Waiting} = 0 \lor \sigma.\#\textsc{Alive} - 1 > \sigma.\#\textsc{Waiting}}{(\texttt{Finish}, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \#\textsc{Alive}{-}{-}, \epsilon)}$$

Deadlock

Fork

$$(\texttt{fork}\ e, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \#\textsc{Alive}{+}{+}, (e; \texttt{Finish}))$$

Acq-Succ

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{false}}{(\texttt{Acq}\ lk, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \textsc{Heap}[lk \leftarrow \texttt{true}], \epsilon)}$$

Acq-Wait

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{true} \qquad \sigma.\#\textsc{Alive} > \sigma.\#\textsc{Waiting} + 1}{(\texttt{Acq}\ lk, \sigma) \rightarrow_{\mathsf{h}} (\texttt{WAIT}\ lk, \sigma : \#\textsc{Waiting}{+}{+}, \epsilon)}$$

Wait-Block

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{true}}{(\texttt{WAIT}\ lk, \sigma) \rightarrow_{\mathsf{h}} (\texttt{WAIT}\ lk, \sigma, \epsilon)}$$

Finish

$$\frac{\sigma.\#\textsc{Waiting} = 0 \vee \sigma.\#\textsc{Alive} - 1 > \sigma.\#\textsc{Waiting}}{(\texttt{Finish}, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \#\textsc{Alive}{-}{-}, \epsilon)}$$

Deadlock

Fork

$$(\texttt{fork}\ e, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \#\textsc{Alive}{+}{+}, (e; \texttt{Finish}))$$

Acq-Succ

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{false}}{(\texttt{Acq}\ lk, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \textsc{Heap}[lk \leftarrow \texttt{true}], \epsilon)}$$

Acq-Wait

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{true} \qquad \sigma.\#\textsc{Alive} > \sigma.\#\textsc{Waiting} + 1}{(\texttt{Acq}\ lk, \sigma) \rightarrow_{\mathsf{h}} (\texttt{WAIT}\ lk, \sigma : \#\textsc{Waiting}{+}{+}, \epsilon)}$$

Wait-Block

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{true}}{(\texttt{WAIT}\ lk, \sigma) \rightarrow_{\mathsf{h}} (\texttt{WAIT}\ lk, \sigma, \epsilon)}$$

Wait-Acq

$$\frac{\sigma.\textsc{Heap}(lk) = \texttt{false}}{(\texttt{WAIT}\ lk, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \textsc{Heap}[lk \leftarrow \texttt{true}] : \#\textsc{Waiting}{-}{-}, \epsilon)}$$

Finish

$$\frac{\sigma.\#\textsc{Waiting} = 0 \lor \sigma.\#\textsc{Alive} - 1 > \sigma.\#\textsc{Waiting}}{(\texttt{Finish}, \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \#\textsc{Alive}{-}{-}, \epsilon)}$$

Agenda

Problem: classic deadlock-freedom specs are too restrictive

Solution: Parameterize modules by client-specific deadlock-freedom argument

Application: Futexes (Compare-and-sleep)

Outlook

Our solution

- Obligations defined on top of the #Alive and #Waiting counters

$$\mathsf{obs}(O) \Longleftrightarrow_{\mathcal{N}_{\text{ACQ}}} \mathsf{obs}(O \uplus \{\!\!\{\lambda\}\!\!\}) * \mathsf{ob}(\lambda)$$

- Deadlock-freedom argument is passed by the client to the blocking module
    - Whenever the module has to block, the client has to show ob(λ) < obs(O)
- Argument phrased in terms of obligations
- Client-managed obligations!

```
// obs(∅)
Acq x                                    ;
// obs(∅)
Finish;
```

```
// obs(∅)
Acq x (fun () -> assert false)    ;
// obs(∅)
Finish;
```

```
// obs(∅)
Acq x (fun () -> assert false) ();
// obs(∅)
Finish;
```

Motivating example

Revisited

```
CreateOblig 0;
lock   x  = NewLock();
bool*  f  = Alloc(true);
```

“If x is true, there is an obligation”

```
// obs(∅)
Acq x (“by inv”) ());
// obs(∅)
f := false;
Finish
```

```
// obs({0})
if !f then exit 0;
// Critical section
Rel x (fun () -> DropOblig(0));
Finish
```

```
// obs(∅)
Acq x ("by inv”) (CreateOblig(0));
// obs({0})
Rel x (DropOblig(1));
// obs(∅)
Finish
```

Client-provided deadlock-freedom argument

$$\frac{P \;{}_{\top}\!\Rrightarrow_{\bot}\; \exists v.\, \ell \mapsto \texttt{true} * \begin{pmatrix} (\ell \mapsto \texttt{true} \;{}_{\bot}\!\Rrightarrow\!\!\ast_{\top}\; P) \\ \vee\, (\ell \mapsto \texttt{false} \;{}_{\bot}\!\Rrightarrow\!\!\ast_{\top}\; Q) \end{pmatrix}}{\{P\}\ \texttt{Rel}(\ell)\ \{Q\}}$$

Cf. logically atomic triples [Jung et al. 2015], (total) atomic triples [D'Osualdo et al. 2021]

Client-provided deadlock-freedom argument

$$\{\mathrm{obs}(\emptyset)\}\ \texttt{Finish}\ \{\mathrm{True}\}$$

$$\frac{\{\mathrm{obs}(O') * P\}\ \texttt{e;Finish}\ \{\mathrm{obs}(\emptyset)\}}{\{\mathrm{obs}(O \uplus O') * P\}\ \texttt{fork}(e)\ \{\mathrm{obs}(O)\}}$$

$$\frac{\begin{array}{c} P\ {}_{\top}{\Rrightarrow}_{\bot}\ \exists v.\ \ell \mapsto \texttt{true} * \\ \left(\begin{array}{l} (\ell \mapsto \texttt{true}\ {}_{\bot}{\equiv\!\!*}_{\top}\ P) \\ \vee\ (\ell \mapsto \texttt{false}\ {}_{\bot}{\equiv\!\!*}_{\top}\ Q) \end{array}\right) \end{array}}{\{P\}\ \texttt{Rel}(\ell)\ \{Q\}}$$

Cf. logically atomic triples [Jung et al. 2015], (total) atomic triples [D'Osualdo et al. 2021]

Client-provided deadlock-freedom argument

$$\{\mathrm{obs}(\emptyset)\}\ \texttt{Finish}\ \{\mathrm{True}\}$$

$$\frac{\{\mathrm{obs}(O') * P\}\ \texttt{e;Finish}\ \{\mathrm{obs}(\emptyset)\}}{\{\mathrm{obs}(O \uplus O') * P\}\ \texttt{fork}(e)\ \{\mathrm{obs}(O)\}}$$

$$\frac{\begin{array}{c} P\ {}_{\top}{\Rrightarrow}_{\bot}\ \exists v.\, \ell \mapsto v * \\ \left(\begin{array}{l} (v = \texttt{true} \wedge \exists \lambda \prec O.\, \mathrm{ob}(\lambda) * (\mathrm{ob}(\lambda) * \ell \mapsto v\ {}_{\bot}{\Rrightarrow}\!\!*_{\top}\ P) \\ \vee (v = \texttt{false} * (\ell \mapsto \texttt{true} * \mathrm{obs}(O)\ {}_{\bot}{\Rrightarrow}\!\!*_{\top}\ Q)) \end{array}\right) \end{array}}{\{\mathrm{obs}(O) * P\}\ \texttt{Acq}(\ell)\ \{Q\}}$$

$$\frac{\begin{array}{c} P\ {}_{\top}{\Rrightarrow}_{\bot}\ \exists v.\, \ell \mapsto \texttt{true} * \\ \left(\begin{array}{l} (\ell \mapsto \texttt{true}\ {}_{\bot}{\Rrightarrow}\!\!*_{\top}\ P) \\ \vee (\ell \mapsto \texttt{false}\ {}_{\bot}{\Rrightarrow}\!\!*_{\top}\ Q) \end{array}\right) \end{array}}{\{P\}\ \texttt{Rel}(\ell)\ \{Q\}}$$

Cf. logically atomic triples [Jung et al. 2015], (total) atomic triples [D'Osualdo et al. 2021]

Agenda

Problem: classic deadlock-freedom specs are too restrictive

Solution: Parameterize modules by client-specific deadlock-freedom argument

Application: Futexes (Compare-and-sleep)

Outlook

Futexes: low-level primitive blocking

Compare-and-sleep*

futex_wait, WaitOnAddress, os_sync_wait_on_address

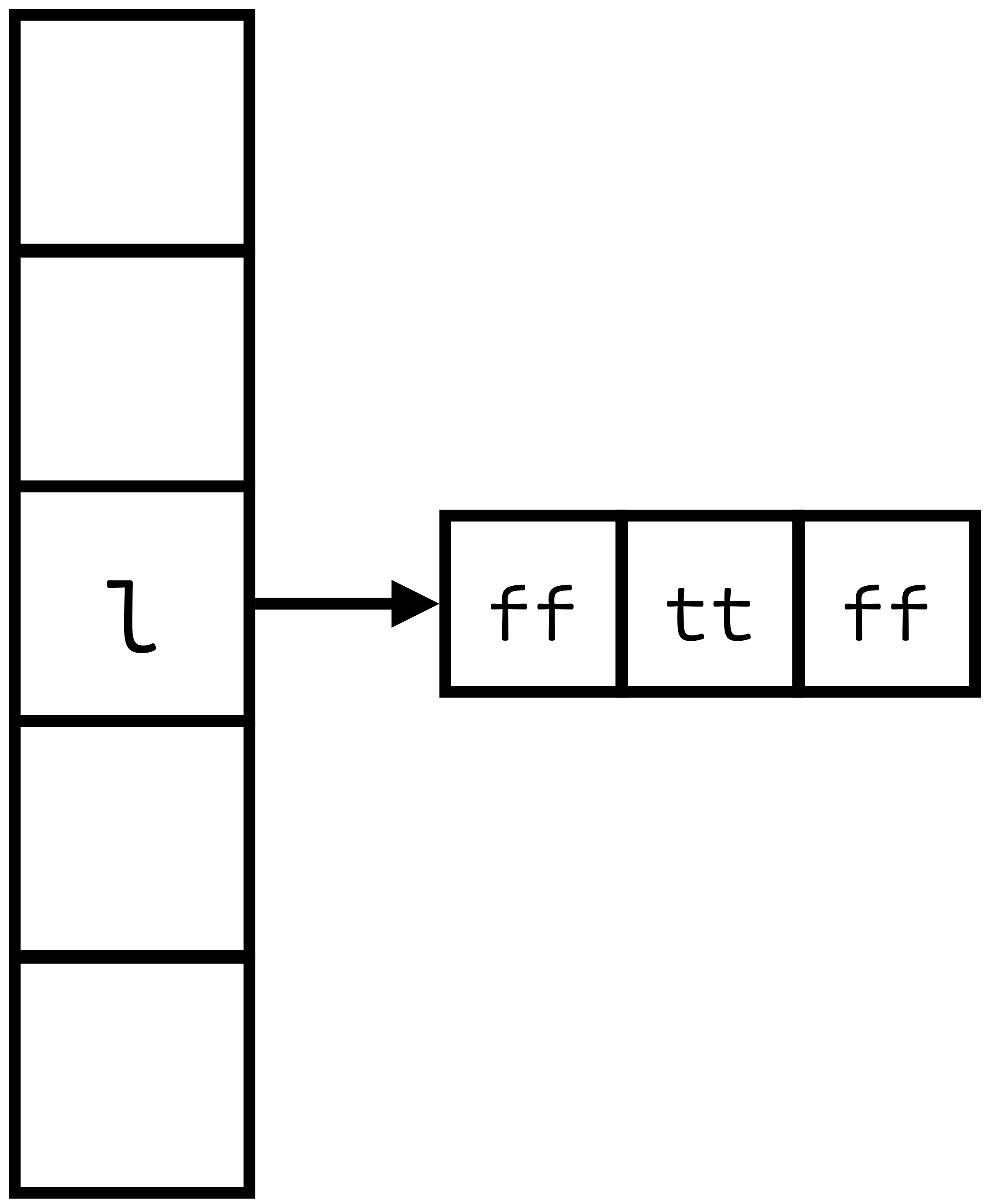
- in-kernel, per-location, list of waiting threads
- `futex_wait l v` adds current thread to the list if `!l = v`
- `futex_wake l` wakes one waiting thread, if there are any
- spurious wakeups are possible

***much simplified: only wake-one, no thread priorities, only same address space, ...**

Compare-and-sleep

How we model the kernel-side

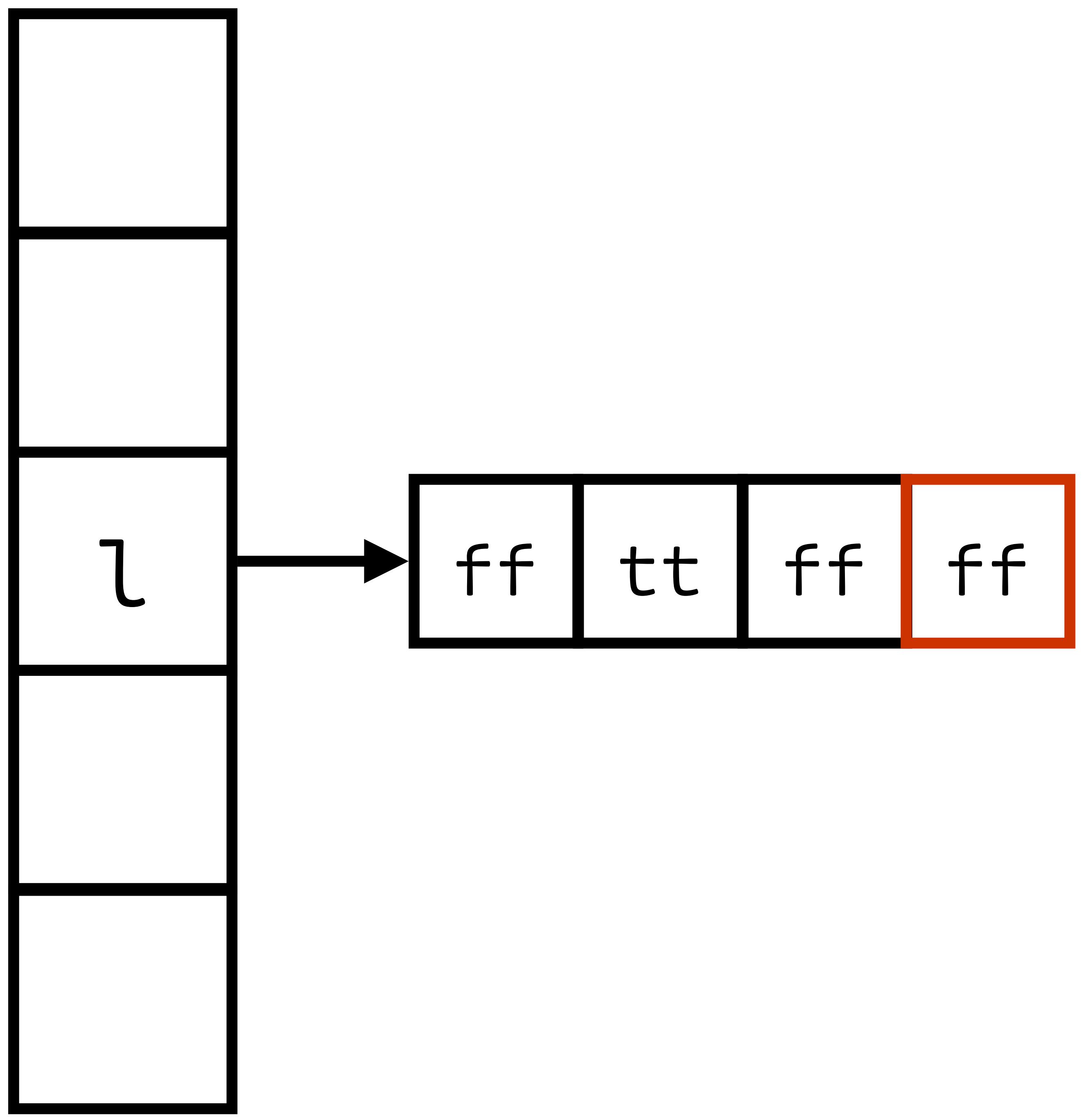
going to sleep:



Compare-and-sleep

How we model the kernel-side

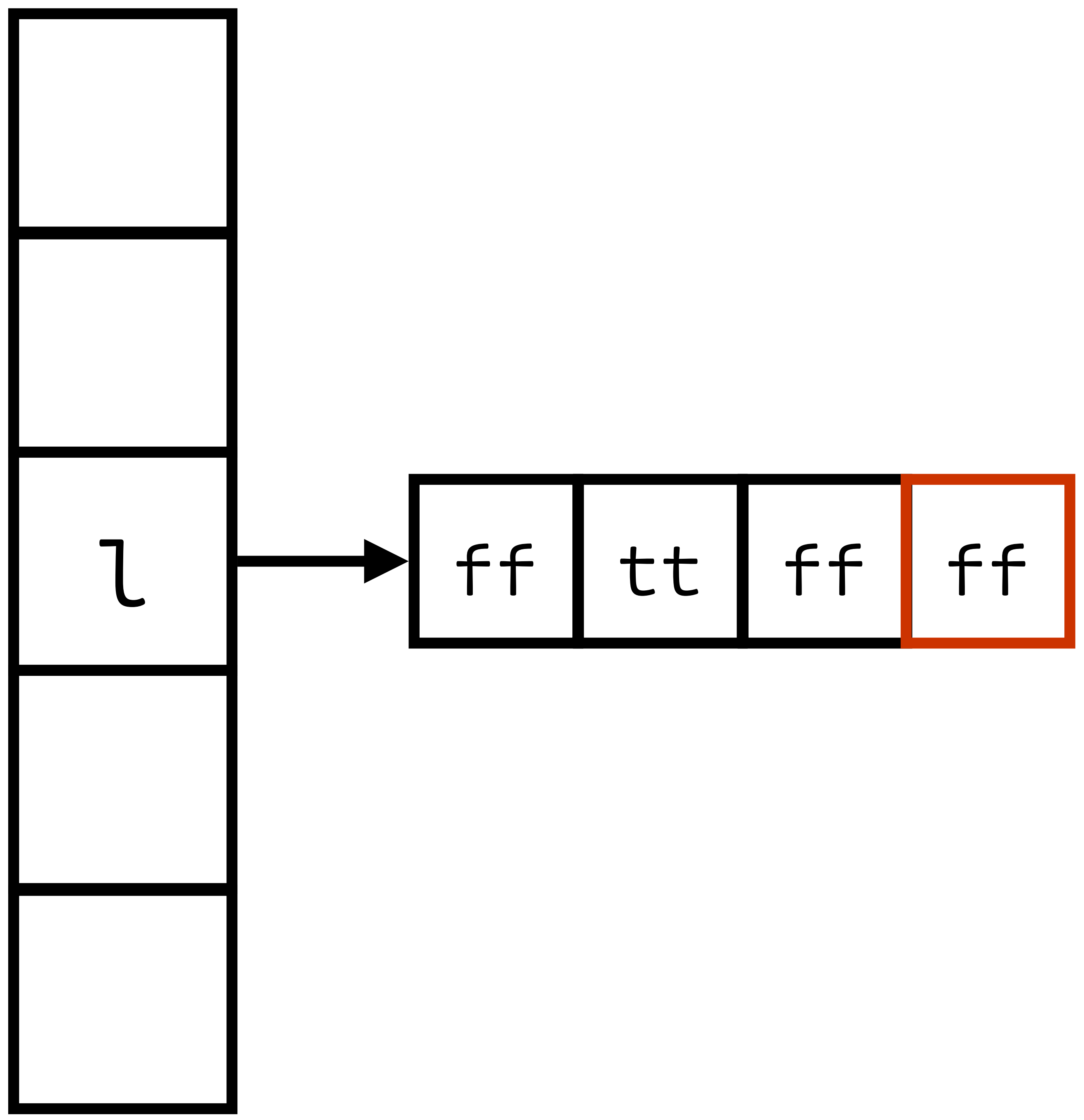
going to sleep:



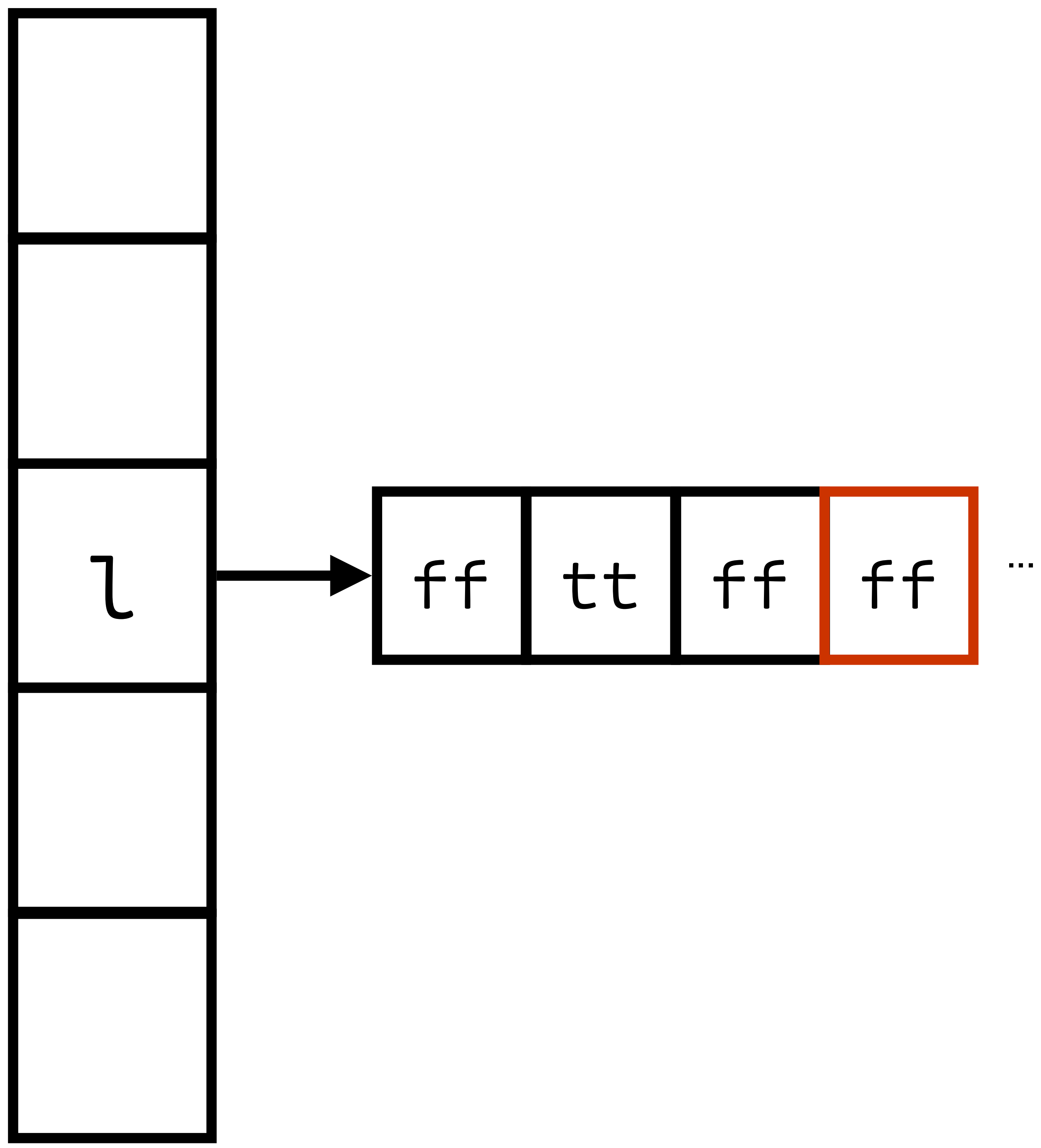
Compare-and-sleep

How we model the kernel-side

going to sleep:



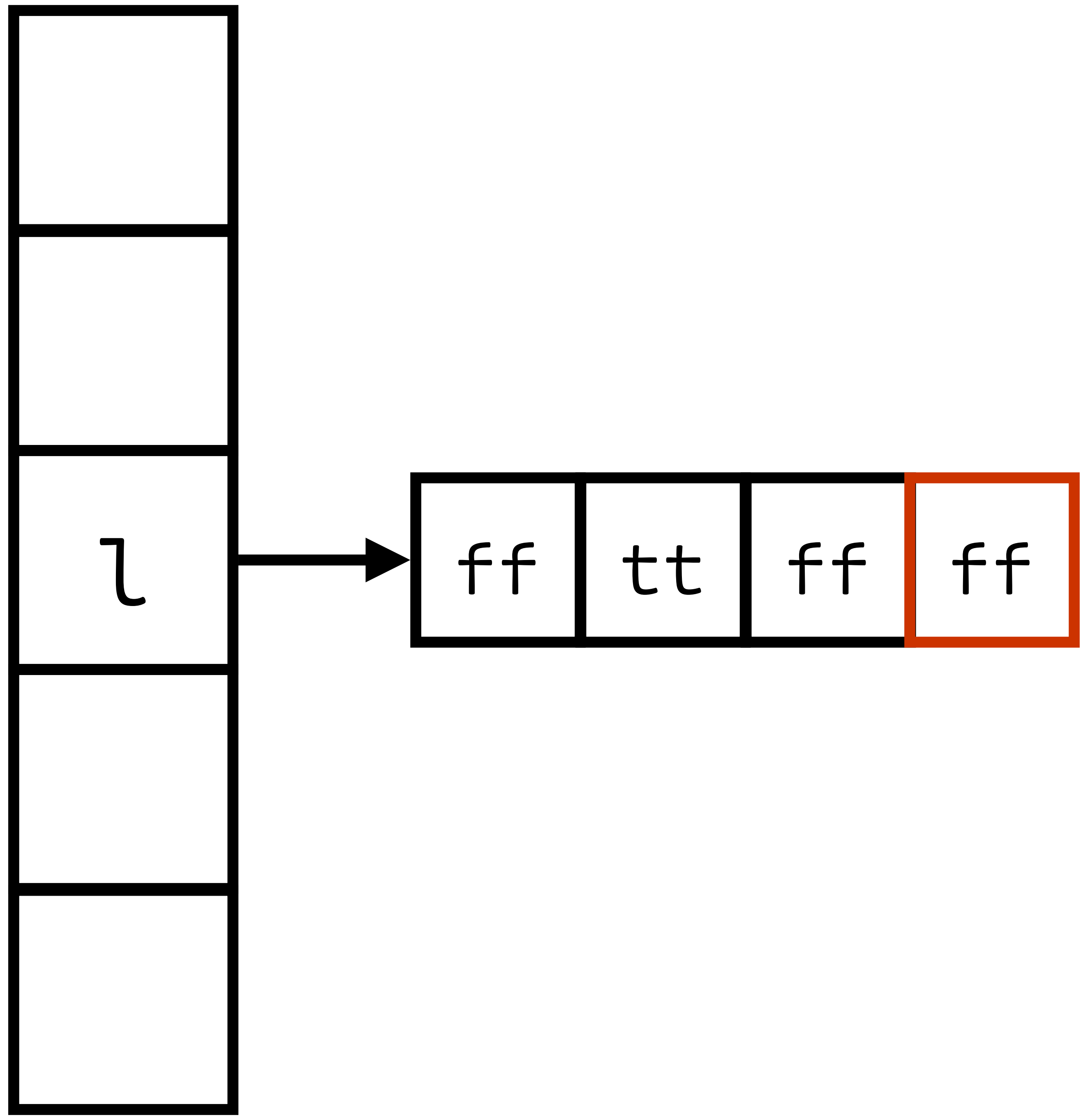
waking up:



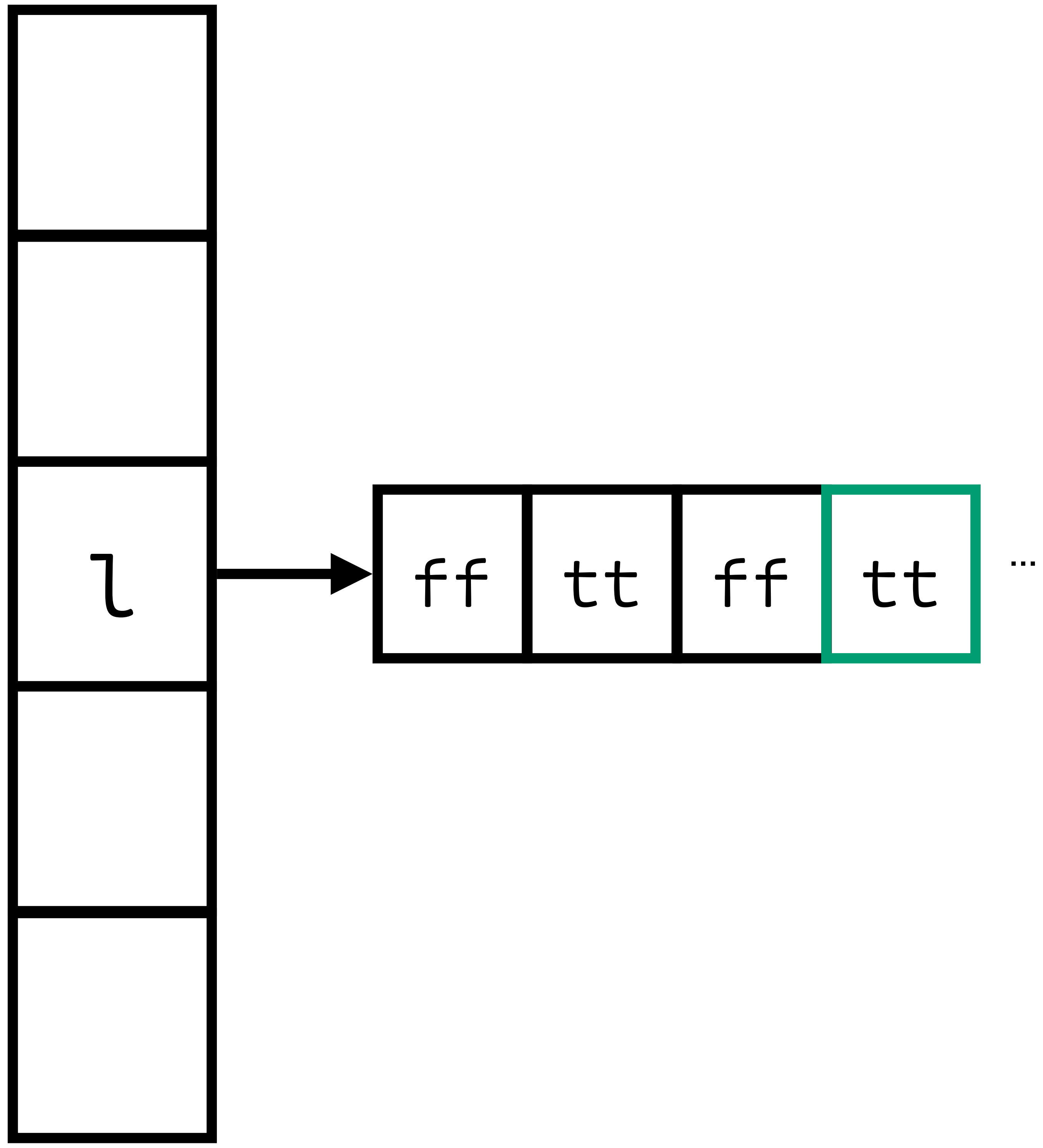
Compare-and-sleep

How we model the kernel-side

going to sleep:



waking up:



Futex step rules

Alloc

$$\frac{\ell \notin \mathrm{dom}(\sigma.\textsc{Heap})}{(\texttt{ref}\ v, \sigma) \rightarrow_{\mathsf{h}} (\ell, \sigma : \textsc{Heap}[\ell \leftarrow v] : \textsc{FutexM}[\ell \leftarrow [\,]], \epsilon)}$$

Futex step rules

ALLOC

$$\frac{\ell \notin \mathrm{dom}(\sigma.\textsc{Heap})}{(\texttt{ref}\; v, \sigma) \rightarrow_{\mathsf{h}} (\ell, \sigma : \textsc{Heap}[\ell \leftarrow v] : \textsc{FutexM}[\ell \leftarrow [\,]], \epsilon)}$$

FUTEXWAITABORT

$$\frac{\sigma.\textsc{Heap}(\ell) = v' \qquad v \neq v'}{(\texttt{futex\_wait}\; \ell\; v, \sigma) \rightarrow_{\mathsf{h}} (\textsc{eagain}, \sigma, \epsilon)}$$

Futex step rules

Alloc

$$\frac{\ell \notin \mathrm{dom}(\sigma.\textsc{Heap})}{(\texttt{ref}\ v, \sigma) \rightarrow_{\mathsf{h}} (\ell, \sigma : \textsc{Heap}[\ell \leftarrow v] : \textsc{FutexM}[\ell \leftarrow [\,]], \epsilon)}$$

FutexWaitAbort

$$\frac{\sigma.\textsc{Heap}(\ell) = v' \qquad v \neq v'}{(\texttt{futex\_wait}\ \ell\ v, \sigma) \rightarrow_{\mathsf{h}} (\textsc{eagain}, \sigma, \epsilon)}$$

FutexWaitWait

$$\frac{\begin{array}{c}\sigma.\textsc{Heap}(\ell) = v \\ \sigma.\#\textsc{Alive} > \sigma.\#\textsc{Waiting} + 1 \qquad \sigma.\textsc{FutexM}(\ell) = B \qquad n = \mathrm{length}(B)\end{array}}{(\texttt{futex\_wait}\ \ell\ v, \sigma) \rightarrow_{\mathsf{h}} (\texttt{WAIT}(\ell, n), \sigma : \textsc{FutexM}[\ell \leftarrow B +\!\!+ [\texttt{false}]] : \#\textsc{Waiting}{+}{+}, \epsilon)}$$

Futex step rules

$$\textsc{Alloc}\quad \frac{\ell \notin \mathrm{dom}(\sigma.\textsc{Heap})}{(\texttt{ref}\ v, \sigma) \rightarrow_{\mathsf{h}} (\ell, \sigma : \textsc{Heap}[\ell \leftarrow v] : \textsc{FutexM}[\ell \leftarrow [\,]], \epsilon)}$$

$$\textsc{FutexWaitAbort}\quad \frac{\sigma.\textsc{Heap}(\ell) = v' \qquad v \neq v'}{(\texttt{futex\_wait}\ \ell\ v, \sigma) \rightarrow_{\mathsf{h}} (\textsc{eagain}, \sigma, \epsilon)}$$

$$\textsc{FutexWaitWait}\quad \frac{\begin{array}{c}\sigma.\textsc{Heap}(\ell) = v \\ \sigma.\#\textsc{Alive} > \sigma.\#\textsc{Waiting} + 1 \qquad \sigma.\textsc{FutexM}(\ell) = B \qquad n = \mathrm{length}(B)\end{array}}{(\texttt{futex\_wait}\ \ell\ v, \sigma) \rightarrow_{\mathsf{h}} (\texttt{WAIT}(\ell, n), \sigma : \textsc{FutexM}[\ell \leftarrow B \mathbin{+\!\!+} [\texttt{false}]] : \#\textsc{Waiting}{+}{+}, \epsilon)}$$

$$\textsc{WaitWait}\quad \frac{\sigma.\textsc{FutexM}(\ell)[n] = \texttt{false}}{(\texttt{WAIT}(\ell, n), \sigma) \rightarrow_{\mathsf{h}} (\texttt{WAIT}(\ell, n), \sigma, \epsilon)}$$

Futex step rules

ALLOC

$$\frac{\ell \notin \text{dom}(\sigma.\textsc{Heap})}{(\texttt{ref}\ v, \sigma) \to_{\mathsf{h}} (\ell, \sigma : \textsc{Heap}[\ell \leftarrow v] : \textsc{FutexM}[\ell \leftarrow [\,]], \epsilon)}$$

FUTEXWAITABORT

$$\frac{\sigma.\textsc{Heap}(\ell) = v' \qquad v \neq v'}{(\texttt{futex\_wait}\ \ell\ v, \sigma) \to_{\mathsf{h}} (\textsc{eagain}, \sigma, \epsilon)}$$

FUTEXWAITWAIT

$$\frac{\begin{array}{c}\sigma.\textsc{Heap}(\ell) = v \\ \sigma.\#\textsc{Alive} > \sigma.\#\textsc{Waiting} + 1 \qquad \sigma.\textsc{FutexM}(\ell) = B \qquad n = \text{length}(B)\end{array}}{(\texttt{futex\_wait}\ \ell\ v, \sigma) \to_{\mathsf{h}} (\texttt{WAIT}(\ell, n), \sigma : \textsc{FutexM}[\ell \leftarrow B +\!\!+ [\texttt{false}]] : \#\textsc{Waiting}{+}{+}, \epsilon)}$$

WAITWAIT

$$\frac{\sigma.\textsc{FutexM}(\ell)[n] = \texttt{false}}{(\texttt{WAIT}(\ell, n), \sigma) \to_{\mathsf{h}} (\texttt{WAIT}(\ell, n), \sigma, \epsilon)}$$

WAITWOKEN

$$\frac{\sigma.\textsc{FutexM}(\ell)[n] = \texttt{true}}{(\texttt{WAIT}(\ell, n), \sigma) \to_{\mathsf{h}} ((), \sigma, \epsilon)}$$

Futex step rules

Futex step rules

FutexWakeOne

$$\frac{\sigma.\textsc{FutexM}(\ell)[n] = \texttt{false}}{(\texttt{futex\_wake}\ \ell, \sigma) \rightarrow_{\mathsf{h}} (1, \sigma : \textsc{FutexM}(\ell)[n \leftarrow \texttt{true}] : \#\textsc{Waiting}{-}{-}, \epsilon)}$$

Futex step rules

FutexWakeOne

$$\frac{\sigma.\textsc{FutexM}(\ell)[n] = \texttt{false}}{(\texttt{futex\_wake}\ \ell, \sigma) \rightarrow_{\mathsf{h}} (1, \sigma : \textsc{FutexM}(\ell)[n \leftarrow \texttt{true}] : \#\textsc{Waiting}{-}{-}, \epsilon)}$$

FutexWakeZero

$$\frac{\sigma.\textsc{FutexM} = B \qquad \mathit{true} = \bigwedge_{b \in B} b}{(\texttt{futex\_wake}\ \ell, \sigma) \rightarrow_{\mathsf{h}} (0, \sigma, \epsilon)}$$

Futex step rules

FutexWakeOne

$$\frac{\sigma.\textsc{FutexM}(\ell)[n] = \texttt{false}}{(\texttt{futex\_wake}\ \ell, \sigma) \rightarrow_{\mathsf{h}} (1, \sigma : \textsc{FutexM}(\ell)[n \leftarrow \texttt{true}] : \#\textsc{Waiting}{-}{-}, \epsilon)}$$

FutexWakeZero

$$\frac{\sigma.\textsc{FutexM} = B \qquad \mathit{true} = \bigwedge_{b \in B} b}{(\texttt{futex\_wake}\ \ell, \sigma) \rightarrow_{\mathsf{h}} (0, \sigma, \epsilon)}$$

WaitSpuriousWake

$$\frac{\sigma.\textsc{FutexM}(\ell)[n] = \texttt{false}}{(\texttt{WAIT}(\ell, n), \sigma) \rightarrow_{\mathsf{h}} ((), \sigma : \textsc{FutexM}(\ell)[n \leftarrow \texttt{true}] : \#\textsc{Waiting}{-}{-}, \epsilon)}$$

Futex step rules

$$
\frac{
\begin{array}{c}
P \;{}_{\top}\!\Rrightarrow_{\bot}\; \exists v', q, B.\; \ell \overset{q}{\mapsto} v' * \mathsf{futex}(\ell, B) * \\
\left(\begin{array}{l}
\left(v \neq v' * (\ell \overset{q}{\mapsto} v' * \mathsf{futex}(\ell, B) \;{}_{\bot}\!\Rrightarrow\!\!\ast_{\top}\; Q(\textsc{eagain}))\right) \\
\vee \left(v = v' * (\qquad (\ell \overset{q}{\mapsto} v' * \mathsf{futex}(\ell, B \mathbin{++} [\mathit{false}]) \qquad {}_{\bot}\!\Rrightarrow\!\!\ast_{\top}\; R(\mathsf{len}(B))))\right)
\end{array} \right)
\end{array}
}{
\{P\}\; \mathtt{futex\_wait}\; \ell\; v\; \{u.\, Q(u)\}
}
$$

Futex proof rules

$$
\frac{
\begin{array}{c}
P \mathrel{{}_{\top}{\Rrightarrow}_{\bot}} \exists v', q, B.\ \ell \overset{q}{\mapsto} v' * \mathrm{futex}(\ell, B) * \\
\left(\begin{array}{l}
\big(v \neq v' * (\ell \overset{q}{\mapsto} v' * \mathrm{futex}(\ell, B) \mathrel{{}_{\bot}{\Rrightarrow\!\!\ast}_{\top}} Q(\text{EAGAIN}))\big) \\
\lor \big(v = v' * (\quad (\ell \overset{q}{\mapsto} v' * \mathrm{futex}(\ell, B +\!\!+ [\mathit{false}]) \quad \mathrel{{}_{\bot}{\Rrightarrow\!\!\ast}_{\top}} R(\mathrm{len}(B))))\big)
\end{array}\right) \\
R(n) \mathrel{{}_{\top}{\Rrightarrow}_{\bot}} \exists B_1, b, B_2.\ n = \mathrm{len}(B_1) * \mathrm{futex}(\ell, B_1 +\!\!+ [b] +\!\!+ B_2) * \\
\left(\begin{array}{l}
\big(b = \mathit{false} \rightarrow \quad (\quad \mathrm{futex}(\ell, B_1 +\!\!+ [b] +\!\!+ B_2) \mathrel{{}_{\bot}{\Rrightarrow\!\!\ast}_{\top}} R(n))\big) \\
\land \big(b = \mathit{true} \rightarrow (\mathrm{futex}(B_1 +\!\!+ [b] +\!\!+ B_2) \mathrel{{}_{\bot}{\Rrightarrow\!\!\ast}_{\top}} Q(0))\big) \\
\land \big(b = \mathit{false} \rightarrow \quad (\quad \mathrm{futex}(B_1 +\!\!+ [\mathit{true}] +\!\!+ B_2) \mathrel{{}_{\bot}{\Rrightarrow\!\!\ast}_{\top}} Q(0))\big)
\end{array}\right)
\end{array}
}{
\{P\}\ \texttt{futex\_wait}\ \ell\ v\ \{u.\ Q(u)\}
}
$$

Futex proof rules

$$\frac{\begin{array}{c} P \;{}_{\top}\!\Rrightarrow_{\bot}\; \exists v', q, B.\, \ell \overset{q}{\mapsto} v' * \mathrm{futex}(\ell, B) * \\ \left(\begin{array}{l} \left(v \neq v' * (\ell \overset{q}{\mapsto} v' * \mathrm{futex}(\ell, B) \;{}_{\bot}\!\Rrightarrow\!\!*_{\top}\; Q(\textsc{eagain}))\right) \\ \vee \left(v = v' * (\qquad (\ell \overset{q}{\mapsto} v' * \mathrm{futex}(\ell, B \mathbin{++} [\mathit{false}]) \qquad {}_{\bot}\!\Rrightarrow\!\!*_{\top}\; R(\mathrm{len}(B))))\right) \end{array} \right) \\ R(n) \;{}_{\top}\!\Rrightarrow_{\bot}\; \exists B_1, b, B_2.\, n = \mathrm{len}(B_1) * \mathrm{futex}(\ell, B_1 \mathbin{++} [b] \mathbin{++} B_2) * \\ \left(\begin{array}{l} \left(b = \mathit{false} \to \qquad (\qquad \mathrm{futex}(\ell, B_1 \mathbin{++} [b] \mathbin{++} B_2) \;{}_{\bot}\!\Rrightarrow\!\!*_{\top}\; R(n))\right) \\ \wedge \left(b = \mathit{true} \to (\mathrm{futex}(B_1 \mathbin{++} [b] \mathbin{++} B_2) \;{}_{\bot}\!\Rrightarrow\!\!*_{\top}\; Q(0))\right) \\ \wedge \left(b = \mathit{false} \to \qquad (\qquad \mathrm{futex}(B_1 \mathbin{++} [\mathit{true}] \mathbin{++} B_2) \;{}_{\bot}\!\Rrightarrow\!\!*_{\top}\; Q(0))\right) \end{array} \right) \end{array}}{\{P\}\ \mathtt{futex\_wait}\ \ell\ v\ \{u.\, Q(u)\}}$$

$$\frac{\begin{array}{c} P \;{}_{\top}\!\Rrightarrow_{\bot}\; \exists B.\, \mathrm{futex}(\ell, B) * \\ \left(\begin{array}{l} (\forall n.\, B[n] = \mathit{false} \to \qquad (\qquad \mathrm{futex}(\ell, B[n \leftarrow \mathit{true}] \;{}_{\bot}\!\Rrightarrow\!\!*_{\top}\; Q(1))) \\ \wedge((\forall n.\, B[n] = \mathit{true}) \to \mathrm{futex}(\ell, B) \;{}_{\bot}\!\Rrightarrow\!\!*_{\top}\; Q(0)) \end{array} \right) \end{array}}{\{P\}\ \mathtt{futex\_wake}\ \ell\ \{u.\, Q(u)\}}$$

Futex proof rules

$$\frac{\begin{array}{c} P \;{}_{\top}{\Rrightarrow}_{\bot}\; \exists v', q, B.\ \ell \overset{q}{\mapsto} v' * \mathsf{futex}(\ell, B) * \\ \left(\begin{array}{l} \left(v \neq v' * (\ell \overset{q}{\mapsto} v' * \mathsf{futex}(\ell, B) \;{}_{\bot}{\Rrightarrow\!\!\ast}_{\top}\; Q(\textsc{eagain}))\right) \\ \lor \left(v = v' * (\exists O.\ \mathsf{obs}(O) * (\ell \overset{q}{\mapsto} v' * \mathsf{futex}(\ell, B +\!\!+ [\mathit{false}]) * \mathsf{wobs}(\ell, \mathsf{len}(B), O) \;{}_{\bot}{\Rrightarrow\!\!\ast}_{\top}\; R(\mathsf{len}(B))))\right) \end{array}\right) \\ R(n) \;{}_{\top}{\Rrightarrow}_{\bot}\; \exists B_1, b, B_2.\ n = \mathsf{len}(B_1) * \mathsf{futex}(\ell, B_1 +\!\!+ [b] +\!\!+ B_2) * \\ \left(\begin{array}{l} \left(b = \mathit{false} \rightarrow \exists O', \lambda \prec O'.\ \mathsf{ob}(\lambda) * \mathsf{wobs}(\ell, n, O') * (\mathsf{ob}(\lambda) * \mathsf{wobs}(\ell, n, O') * \mathsf{futex}(\ell, B_1 +\!\!+ [b] +\!\!+ B_2) \;{}_{\bot}{\Rrightarrow\!\!\ast}_{\top}\; R(n))\right) \\ \land \left(b = \mathit{true} \rightarrow (\mathsf{futex}(B_1 +\!\!+ [b] +\!\!+ B_2) \;{}_{\bot}{\Rrightarrow\!\!\ast}_{\top}\; Q(0))\right) \\ \land \left(b = \mathit{false} \rightarrow \exists O'.\ \mathsf{wobs}(\ell, n, O') * (\mathsf{obs}(O') * \mathsf{futex}(B_1 +\!\!+ [\mathit{true}] +\!\!+ B_2) \;{}_{\bot}{\Rrightarrow\!\!\ast}_{\top}\; Q(0))\right) \end{array}\right) \end{array}}{\{P\}\ \texttt{futex\_wait}\ \ell\ v\ \{u.\ Q(u)\}}$$

$$\frac{\begin{array}{c} P \;{}_{\top}{\Rrightarrow}_{\bot}\; \exists B.\ \mathsf{futex}(\ell, B) * \\ \left(\begin{array}{l} (\forall n.\ B[n] = \mathit{false} \rightarrow \exists O.\ \mathsf{wobs}(\ell, n, O) * (\mathsf{obs}(O) * \mathsf{futex}(\ell, B[n \leftarrow \mathit{true}] \;{}_{\bot}{\Rrightarrow\!\!\ast}_{\top}\; Q(1))) \\ \land((\forall n.\ B[n] = \mathit{true}) \rightarrow \mathsf{futex}(\ell, B) \;{}_{\bot}{\Rrightarrow\!\!\ast}_{\top}\; Q(0)) \end{array}\right) \end{array}}{\{P\}\ \texttt{futex\_wake}\ \ell\ \{u.\ Q(u)\}}$$

Futex proof rules

Agenda

Problem: classic deadlock-freedom specs are too restrictive

Solution: Parameterize modules by client-specific deadlock-freedom argument

Application: Futexes (Compare-and-sleep)

Outlook

Outlook

Locks

Futex

Outlook

Condition variables

Channels

Semaphores

Locks

Futex

Outlook

Condition variables

Channels

Semaphores

Locks

Futex

Busy-waiting

Outlook

- Deadlock-free monitors
- Obligations transfer via channels

Hamin and Jacobs 2018, 2019

Near future: “Futexes are tricky”

Optimized futex mutex

```
class mutex3 {
public:
  mutex() : val(0) {}

  void lock() {
    int c;
    if ((c = cmpxchg(val, 0, 1)) != 0) {
      if (c != 2)
        c = xchg(val, 2);
      while (c != 0) {
        futex_wait(&val, 2);
        c = xchg(val, 2);
      }
    }
  }

  void unlock() {
    if (atomic_dec(val) != 1) {
      val = 0;
      futex_wake(&val, 1);
    }
  }

private:
  int val;
};
```

Drepper 2011

🤘Block on🤘