Ch 13 – Self-Stabilizing Lynch-Welch

The objective is to make the Lynch-Welch algorithm of Ch10 withstand any number of transient faults and and at the same time up to f Byzantine faults.

Algorithm 11 Lynch-Welch pulse synchronization algorithm, the code for a node $v \in V_g$. $\mathcal{S}$ denotes a (to-be-determined) upper bound on $\|\vec{p}_r\|$ for each $r \in \mathbb{N}_{>0}$ and T is the nominal round duration and it needs to be specified how Line 9 is implemented.

1: wait until getH() $= \mathcal{S}$ // $H_w(0) \in [0, \mathcal{S})$ for all $w \in V$
2: **for all** round $r \in \mathbb{N}$ **do**
3:     generate r-th pulse
4:     $h \leftarrow$ getH()
5:     wait until getH() $= h + \vartheta\mathcal{S}$ // all nodes are in round r
6:     broadcast empty message to all nodes (including self)
7:     wait until getH() $= h + (\vartheta^2 + \vartheta)\mathcal{S} + \vartheta d$ // denote this time by $\tau_{v,r}$
// correct nodes' messages should have arrived
8:     **for** each node $w \in V$ **do**
9:         compute $\Delta(w) \in [p_{w,r} - p_{v,r}, p_{w,r} - p_{v,r} + \delta]$
// denote $p_r := \max_{w \in V_g}\{p_{w,r}\}$
10:     **end for**
11:     $S \leftarrow \{\Delta(w) \mid w \in V\}$ (as multiset, i.e., values may repeat)
12:     $\Delta \leftarrow \left(S_v^{(f+1)} + S_v^{(n-f)}\right)/2$
13:     wait until getH() $= h + \Delta + T$
14: **end for**

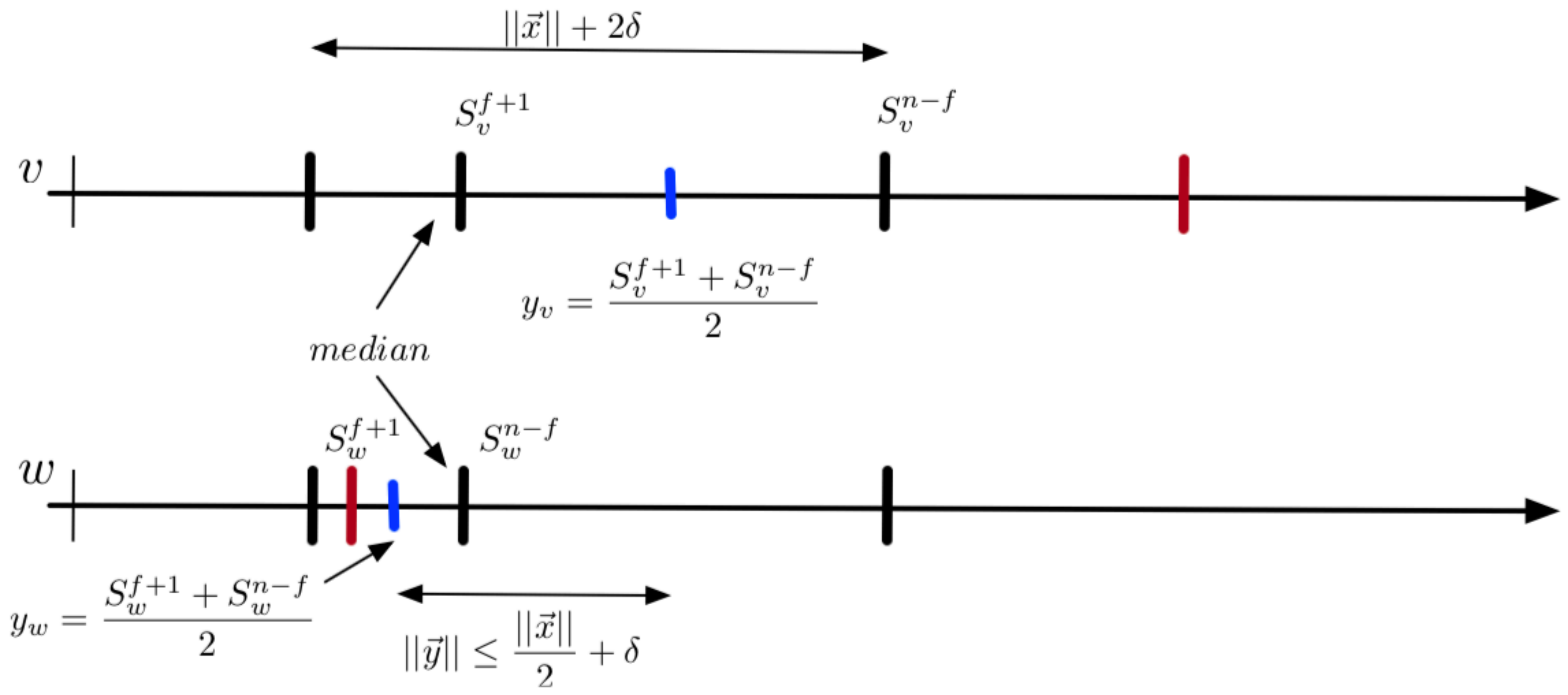
Initial requirements on round execution

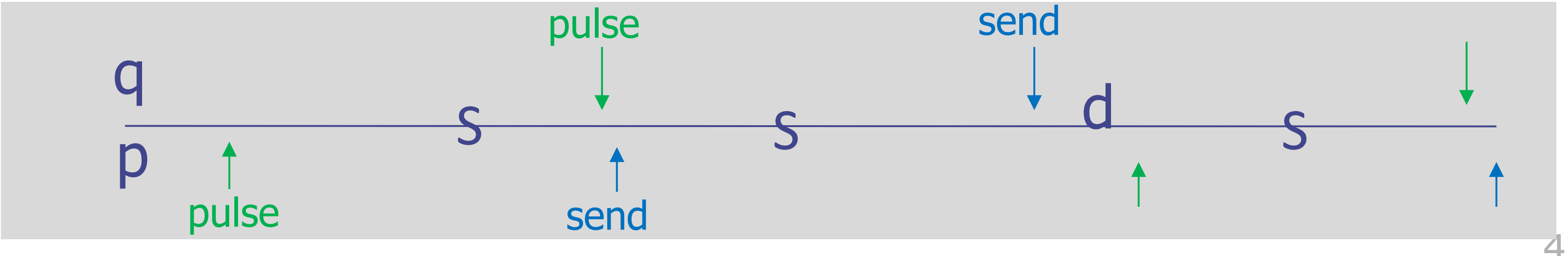
Since we simulate an approximate agreement in a synchronous environment we need to make sure that the following holds:

1) Messages sent by correct nodes in a given round should be received by all correct nodes after they start the current round and before they compute the clock estimates, i.e., during $[p_{v,r}, \tau_{v,r}]$

2) T is large enough to accommodate the adjustments for the next iteration, i.e., $H_v(\tau_{v,r}) \leq H_v(p_{v,r}) + \Delta + T$

4 nodes – 1 faulty and perturbation in values





Self-Stabilizing Lynch-Welch

The objective is to make the Lynch-Welch algorithm of Ch10 withstand any number of transient faults and and at the same time up to f Byzantine faults.

- Main weaknesses of the original algorithm:
  1. assumed pretty strong synchronization
  2. assumed a small initial skew
  3. assumed receiving of at least n-t values from correct nodes in each "window"

Algorithm 16 The loop of Algorithm 11, which is run alongside the local instances of the beat generation algorithm and Algorithm 17. Note that Algorithm 17 may reset the loop for stabilization purposes.

first line removed

while true **do**
    generate pulse   // assume that $r \in \mathbb{N}$ is the pulse index
    $h \leftarrow$ getH()
    wait until getH() $= h + \vartheta \mathcal{S}$ // all nodes are in round r
    broadcast empty message to all nodes (including self)
    wait until getH() $= h + (\vartheta^2 + \vartheta)\mathcal{S} + \vartheta d$
        // denote this time by $\tau_{v,r}$
        // correct nodes' messages should have arrived
    for each node $w \in V$ **do**
        compute $\Delta(w) \in [p_{w,r} - p_{v,r}, p_{w,r} - p_{v,r} + \delta]$
            // denote $p_r := \max_{w \in V_g}\{p_{w,r}\}$
    end for
    $U \leftarrow \{\Delta(w) \mid w \in V\}$ (as multiset, i.e., values may repeat)
    $\Delta \leftarrow \left(U^{(f+1)} + U^{(n-f)}\right)/2$
    wait until getH() $= h + \Delta + T$
end while

we cannot wait for n-t values

To prevent chaos the algorithm will be reset when events do not line up properly

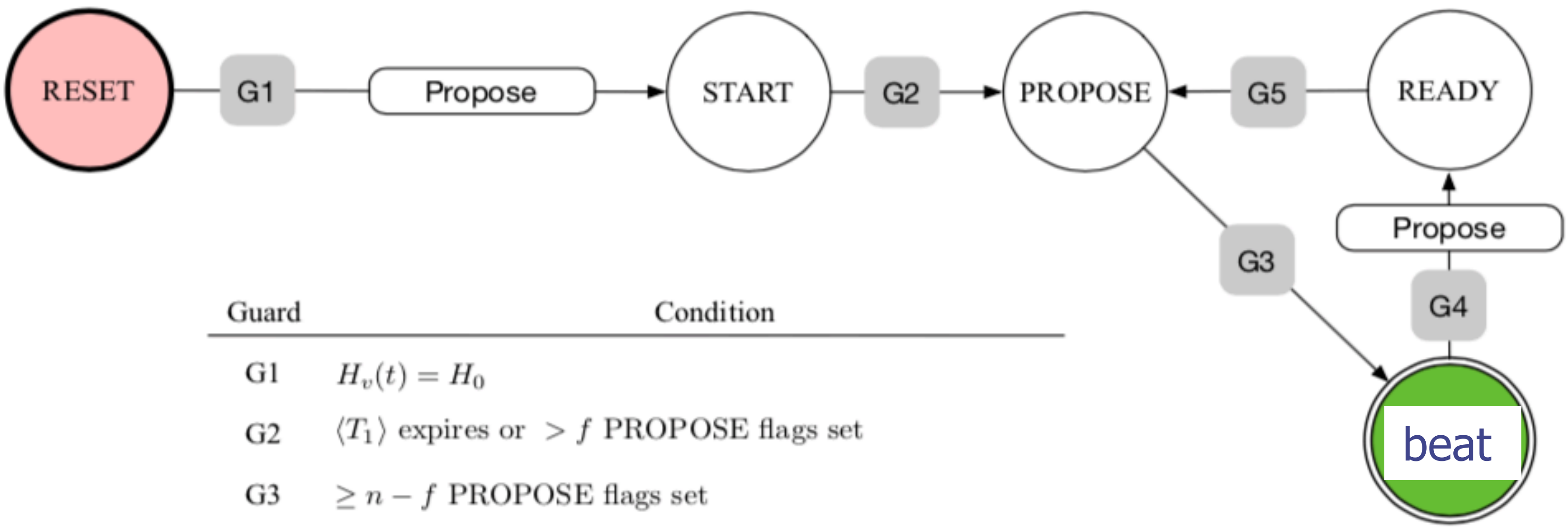
Background Synchronization

We assume: some time after the number of existing faults falls below t, the following can be achieved:

- Every correct node generates an event (**Beat**) at a "regular" period
- All events of non-faulty nodes in each wave of events are within some $\boldsymbol{\sigma}_h$ of each other
- We have lower and upper bounds on the period length between waves of events.

We will show how to obtain that in a later chapter.

Background Synchronization

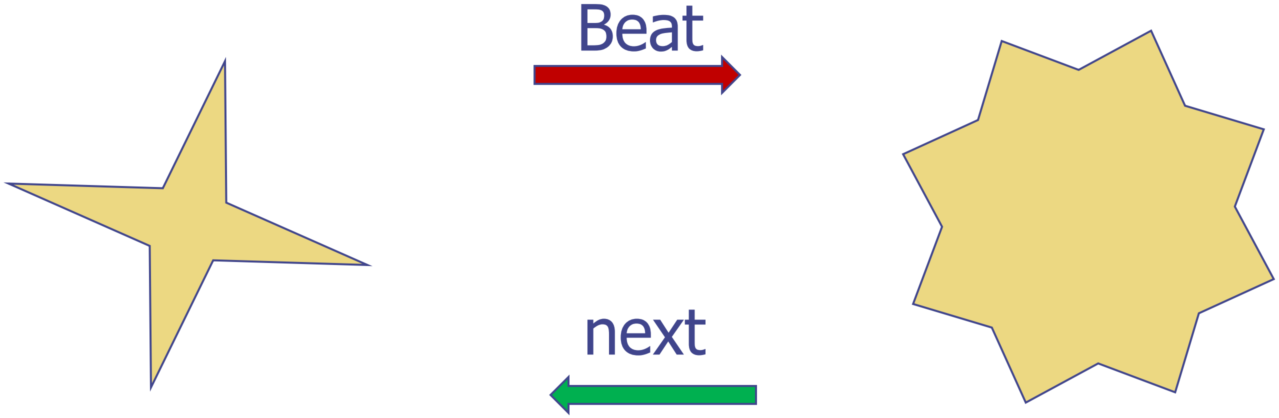


Guard	Condition
G1	$H_v(t) = H_0$
G2	$\langle T_1 \rangle$ expires or $> f$ PROPOSE flags set
G3	$\geq n - f$ PROPOSE flags set
G4	$\langle T_2 \rangle$ expires
G5	$\langle T_3 \rangle$ expires or $> f$ PROPOSE flags set

We will make this state-machine self-stabilizing

Can this replace LW?

Coordinating Two Independent Cycles



- To prevent chaos we need to coordinate the two streams of events
- Our aim is to reduce the skew – waiting for the slow process to produce the desired beat increases the skew.

- We need another approach

Beats and Feedbacks

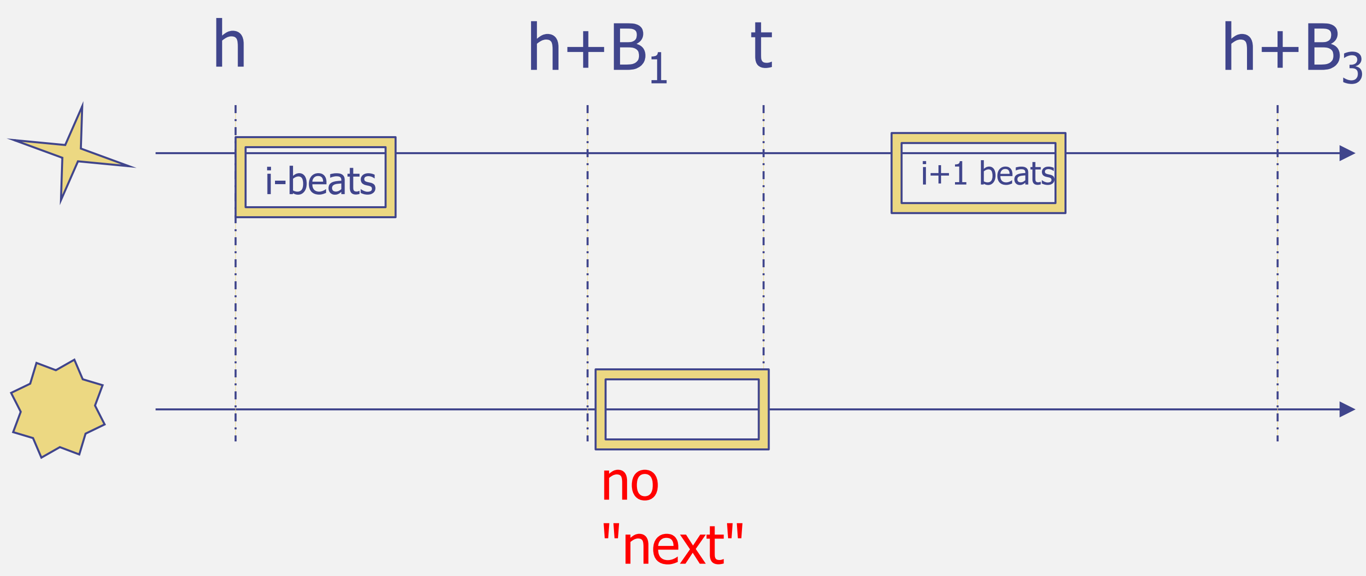
Definition 13.2 (Feedback Mechanism). *Nodes $v \in V_g$ generate* beats *at times $h_{v,i} \in \mathbb{R}$, $i \in \mathbb{N}$, such that for parameters $0 < B_1 < B_2 < B_3 \in \mathbb{R}$ and σ_h (a skew bound) the following properties hold, for all $i \in \mathbb{N}$.*

1. *For all $v, w \in V_g$, we have that $|h_{v,i} - h_{w,i}| \leq \sigma_h$.*

Beats and Feedbacks

Definition 13.2 (Feedback Mechanism). *Nodes $v \in V_g$ generate* beats *at times $h_{v,i} \in \mathbb{R}$, $i \in \mathbb{N}$, such that for parameters $0 < B_1 < B_2 < B_3 \in \mathbb{R}$ and σ_h (a skew bound) the following properties hold, for all $i \in \mathbb{N}$.*

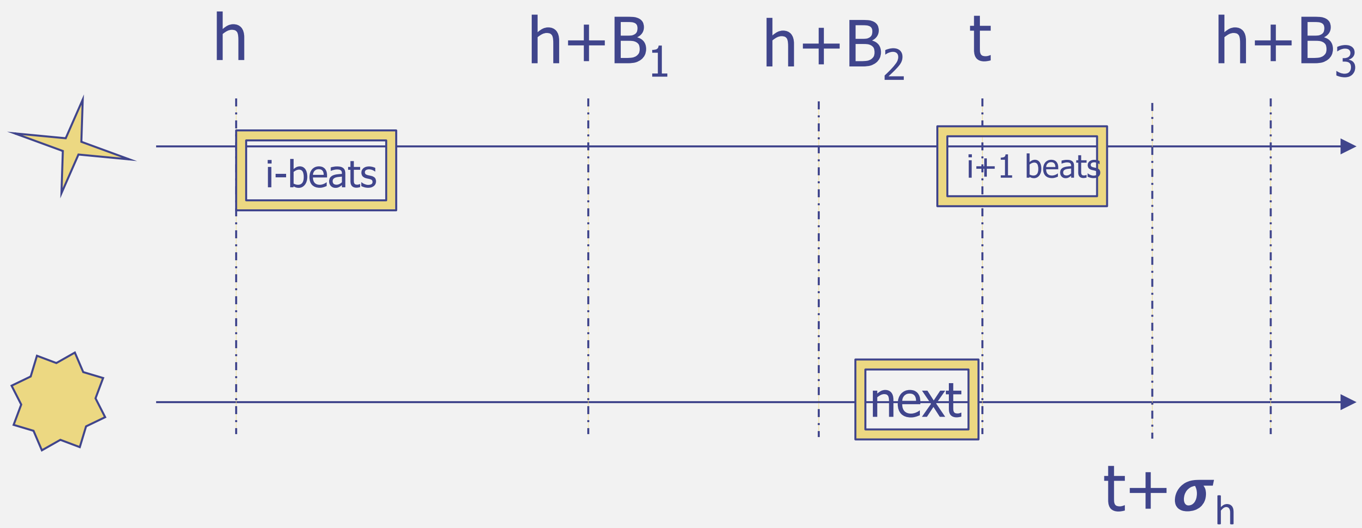
1. *For all $v, w \in V_g$, we have that $|h_{v,i} - h_{w,i}| \leq \sigma_h$.*
2. *If no $v \in V_g$ triggers its NEXT signal during $[\min_{w \in V_g}\{h_{w,i}\} + B_1, t]$ for some $t < \min_{w \in V_g}\{h_{w,i}\} + B_3$, then $\min_{w \in V_g}\{h_{w,i+1}\} > t$.*



Beats and Feedbacks

Definition 13.2 (Feedback Mechanism). *Nodes $v \in V_g$ generate* beats *at times $h_{v,i} \in \mathbb{R}$, $i \in \mathbb{N}$, such that for parameters $0 < B_1 < B_2 < B_3 \in \mathbb{R}$ and σ_h (a skew bound) the following properties hold, for all $i \in \mathbb{N}$.*

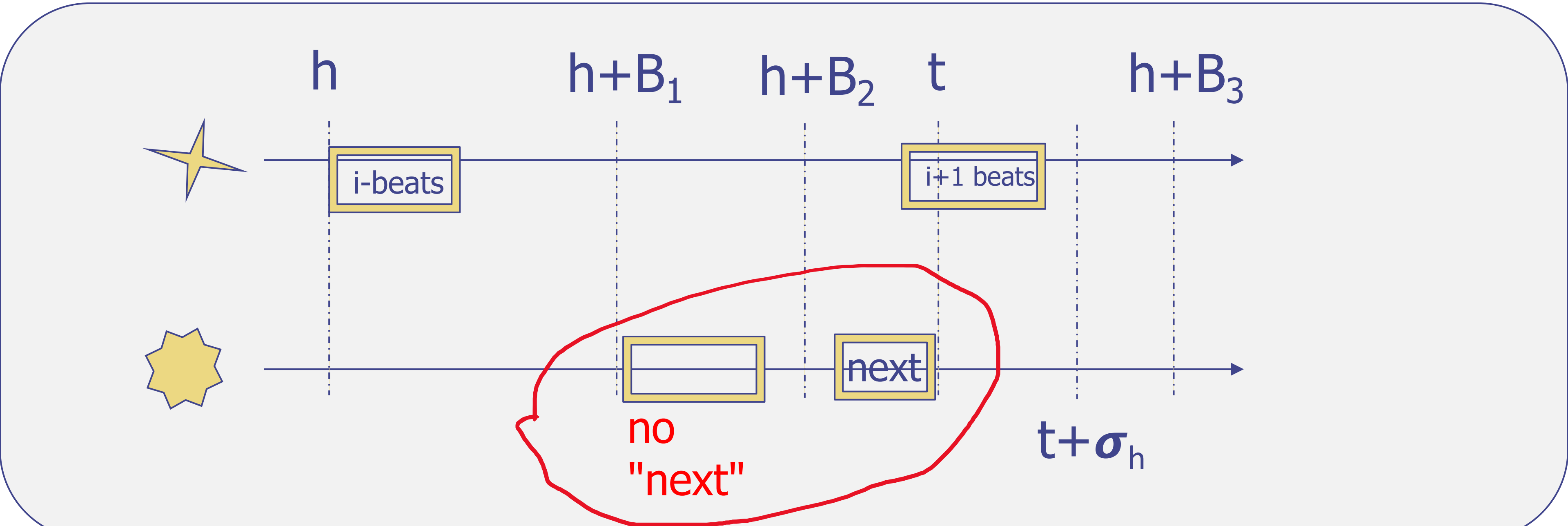
1. *For all $v, w \in V_g$, we have that $|h_{v,i} - h_{w,i}| \le \sigma_h$.*
2. *If no $v \in V_g$ triggers its NEXT signal during $[\min_{w \in V_g}\{h_{w,i}\} + B_1, t]$ for some $t < \min_{w \in V_g}\{h_{w,i}\} + B_3$, then $\min_{w \in V_g}\{h_{w,i+1}\} > t$.*
3. *If all $v \in V_g$ trigger their NEXT signals during $[\min_{w \in V_g}\{h_{w,i}\} + B_2, t]$ for some $t \le \min_{w \in V_g}\{h_{w,i}\} + B_3$, then $\max_{w \in V_g}\{h_{w,i+1}\} \le t + \sigma_h$.*



Beats and Feedbacks

Definition 13.2 (Feedback Mechanism). *Nodes $v \in V_g$ generate* beats *at times $h_{v,i} \in \mathbb{R}$, $i \in \mathbb{N}$, such that for parameters $0 < B_1 < B_2 < B_3 \in \mathbb{R}$ and σ_h (a skew bound) the following properties hold, for all $i \in \mathbb{N}$.*

1. *For all $v, w \in V_g$, we have that $|h_{v,i} - h_{w,i}| \leq \sigma_h$.*
2. *If no $v \in V_g$ triggers its NEXT signal during $[\min_{w \in V_g}\{h_{w,i}\} + B_1, t]$ for some $t < \min_{w \in V_g}\{h_{w,i}\} + B_3$, then $\min_{w \in V_g}\{h_{w,i+1}\} > t$.*
3. *If all $v \in V_g$ trigger their NEXT signals during $[\min_{w \in V_g}\{h_{w,i}\} + B_2, t]$ for some $t \leq \min_{w \in V_g}\{h_{w,i}\} + B_3$, then $\max_{w \in V_g}\{h_{w,i+1}\} \leq t + \sigma_h$.*



Plan

- If events do not lineup – we will reset LW - an extreme measure that we take to converge from faults
- To obtain small skew we should not take such an action when events line up.
- To overcome the extra skew that the synchronization with the beats produce, we reduce the skew prior to that stage. We repeat the LW (approximate agreement) loop for several times in a row. (M iterations)

How to put all of that together?

Recall: Approximate Agreement Algorithm

The algorithm proceeds in rounds.

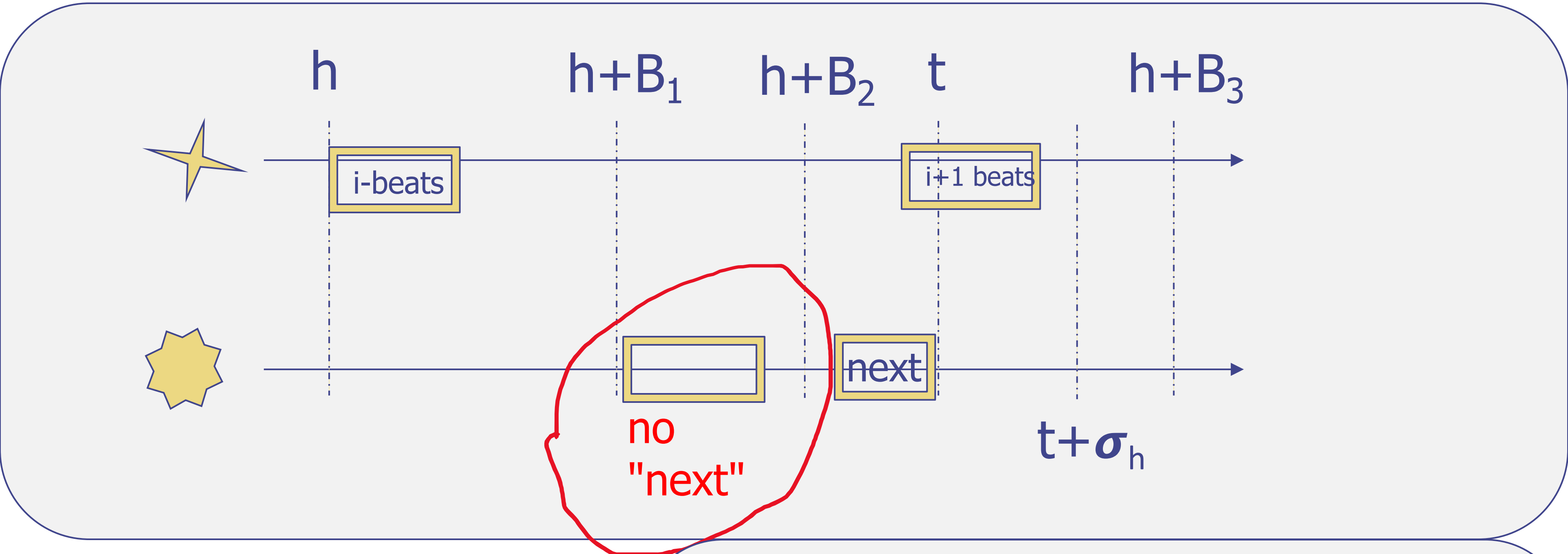
The Basic Iteration:

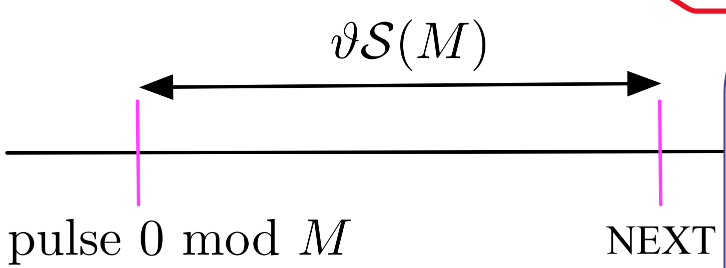
1. send r_v to all.
2. receive $r_{w,v}$, the value sent by w in this round.
   // replace any "missing" value by r_v
3. $S_v := \{r_{w,v}\}$; //ordered set
4. $o_v := (S_v^{(f+1)} + S_v^{(n-f)})/2$; // the (f+1)st and (n-f)-th values in S
5. Return o_v

The initial range is reduced by half at the end of each iteration

We will start with S and end with S(M) after M iterations

Beats and Pulses Alignment





If pulses are clustered we get the "**no next**" window followed by the "**next**" window

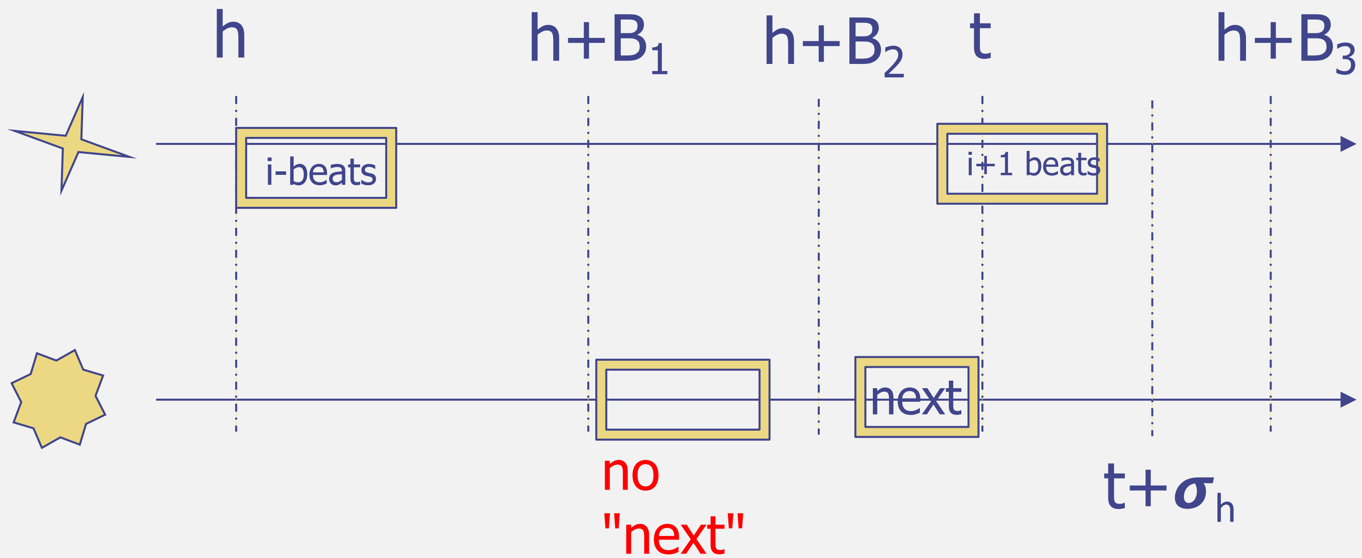
The Meta Algorithm

Algorithm 17 Interface algorithm, actions for node $v \in V_g$ in response to a local event at time t. Runs in parallel to local instances of the beat generation algorithm and Algorithm 16.

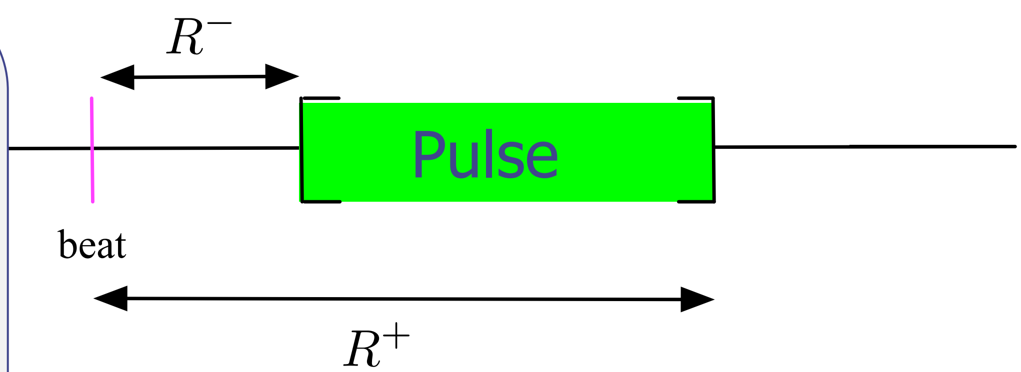
```
                                  ▷ The algorithm maintains local variable i ∈ [M]
if v generates a pulse at time t then
    i := i + 1 mod M
    if i = 0 then
        wait until local time H_v(t) + ϑS(M)
        trigger NEXT signal
    end if
end if
```

- Cycle for M rounds –
- Wait for the skew to pass
- Invoke a NEXT event

Beats and Pulses Alignment



A well aligned **Beat** requires that the next "**Pulse 1**" will be within the "**green**" window



Otherwise – invoke a Reset

Algorithm 17 Interface algorithm, actions for node $v \in V_g$ in response to a local event at time t. Runs in parallel to local instances of the beat generation algorithm and Algorithm 16.

▹ The algorithm maintains local variable $i \in [M]$
if v generates a pulse at time t **then**
    $i := i + 1 \bmod M$
    if $i = 0$ **then**
        wait until local time $H_v(t) + \vartheta S(M)$
        trigger NEXT signal
    end if
end if
if v generates a beat at time t **then**
    if $i \neq 0$ **then**
        ▹ beats should align with every M^{th} pulse, hence reset
        reset(R^+)
    else if Algorithm 16 requires generating a pulse before $H_v(t) + R^-$ **then**
        ▹ reset at pulse time t' to avoid early pulse or message
        reset$(R^+ - (H_v(t') - H_v(t)))$, where t' is the current time
    else if next pulse is not generated by local time $H_v(t) + R^+$ **then**
        ▹ reset to avoid late pulse and
        ▹ start listening for other nodes' pulses on time
        reset(0)
    end if
end if
Function(**reset**(τ))
stop local instance of Algorithm 16
wait for τ local time
$i := 0$
initialize a new local instance of Algorithm 16

---

if v generates a beat at time t **then**

  if $i \neq 0$ **then**

    ▹ beats should align with every M^{th} pulse, hence reset

    reset(R^{+})

Function(reset(τ)**)**

stop local instance of Algorithm 16

wait for τ local time

$i := 0$

initialize a new local instance of Algorithm 16

if v generates a beat at time t **then**

  if $i \neq 0$ **then**

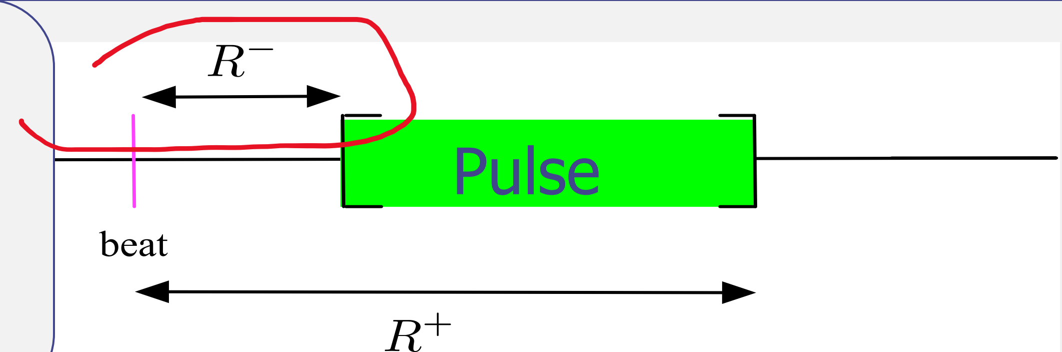
    ▷ beats should align with every M^{th} pulse, hence reset

    reset(R^{+})

  else if Algorithm 16 requires generating a pulse before $H_v(t) + R^{-}$ **then**

    ▷ reset at pulse time t' to avoid early pulse or message

    reset($R^{+} - (H_v(t') - H_v(t))$), where t' is the current time



Function(**reset**(τ))

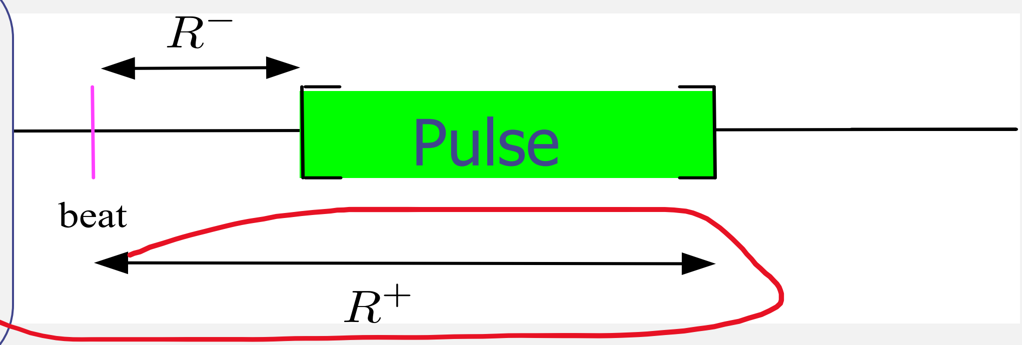
stop local instance of Algorithm 16

wait for τ local time

$i := 0$

initialize a new local instance of Algorithm 16

```
if v generates a beat at time t then
    else if next pulse is not generated by local time H_v(t) + R⁺ then
                                          ▷ reset to avoid late pulse and
                            ▷ start listening for other nodes' pulses on time
        reset(0)
    end if
end if
Function(reset(τ))
stop local instance of Algorithm 16
wait for τ local time
i := 0
initialize a new local instance of Algorithm 16
```



We force an immediate pulse

```
if v generates a beat at time t then
```


```
    else if next pulse is not generated by local time H_v(t) + R⁺ then
                                        ▷ reset to avoid late pulse and
                        ▷ start listening for other nodes' pulses on time
        reset(0)
    end if
end if
Function(reset(τ))
stop local instance of Algorithm 16
wait for τ local time
i := 0
initialize a new local instance of Algorithm 16
```

i=0 and well aligned (green window)

if v generates a beat at time t **then**
    if $i \neq 0$ **then**
        ▹ beats should align with every M^{th} pulse, hence reset
        reset(R^+)
    else if Algorithm 16 requires generating a pulse before $H_v(t) + R^-$ **then**
        ▹ reset at pulse time t' to avoid early pulse or message
        reset($R^+ - (H_v(t') - H_v(t))$), where t' is the current time
    else if next pulse is not generated by local time $H_v(t) + R^+$ **then**
        ▹ reset to avoid late pulse and
        ▹ start listening for other nodes' pulses on time
        reset(0)
    end if
end if
Function(**reset**(τ))
stop local instance of Algorithm 16
wait for τ local time
$i := 0$
initialize a new local instance of Algorithm 16

delay the next pulse

delay the next pulse

force a pulse

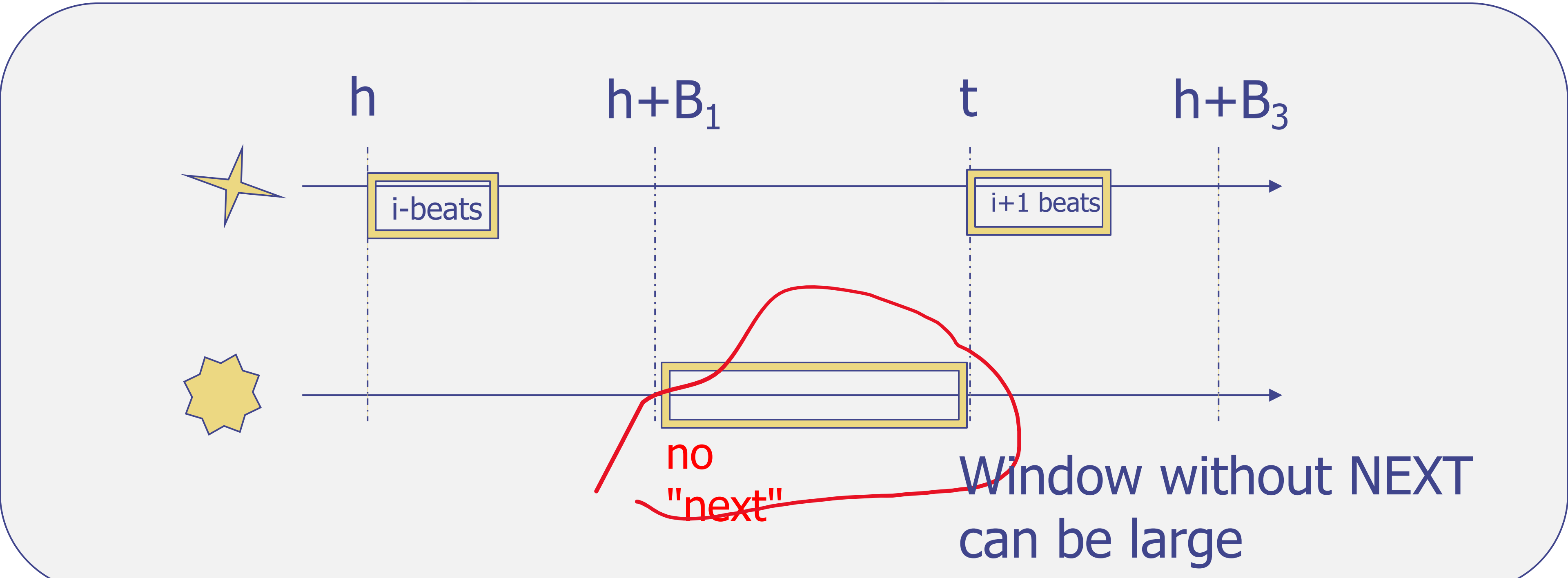
i=0 and well aligned (green window)

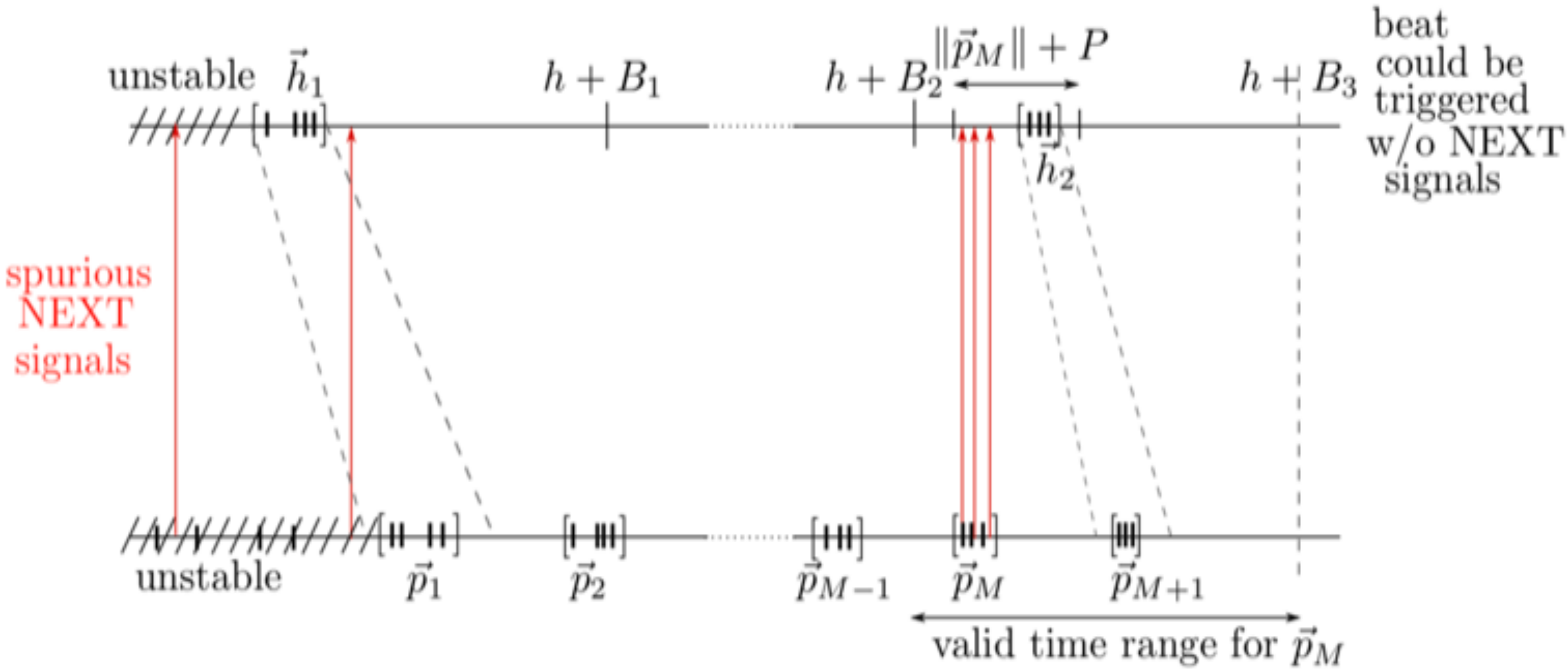
From the pseudocode given in Algorithm 17, it is straightforward to verify that $v \in V_g$ generates a pulse at a local time from $[H_v(h_{v,1}) + R^-, H_v(h_{v,1}) + R^+]$, and does not generate a pulse at a local time from $[H_v(h_{v,1}), H_v(h_{v,1}) + R^-)$.

Beats and Feedbacks

Definition 13.2 (Feedback Mechanism). *Nodes $v \in V_g$ generate* beats *at times $h_{v,i} \in \mathbb{R}$, $i \in \mathbb{N}$, such that for parameters $0 < B_1 < B_2 < B_3 \in \mathbb{R}$ and σ_h (a skew bound) the following properties hold, for all $i \in \mathbb{N}$.*

1. *For all $v, w \in V_g$, we have that $|h_{v,i} - h_{w,i}| \leq \sigma_h$.*
2. *If no $v \in V_g$ triggers its NEXT signal during $[\min_{w \in V_g}\{h_{w,i}\} + B_1, t]$ for some $t < \min_{w \in V_g}\{h_{w,i}\} + B_3$, then $\min_{w \in V_g}\{h_{w,i+1}\} > t$.*




unstable
$\vec{h}_1$
$h + B_1$
$h + B_2$
$\|\vec{p}_M\| + P$
$h + B_3$
beat could be triggered w/o NEXT signals
$\vec{h}_2$
spurious NEXT signals
unstable
$\vec{p}_1$
$\vec{p}_2$
$\vec{p}_{M-1}$
$\vec{p}_M$
$\vec{p}_{M+1}$
valid time range for $\vec{p}_M$