CIS 842: Specification and Verification of Reactive Systems

Lecture SPIN-Soldiers: Soldiers Case Study

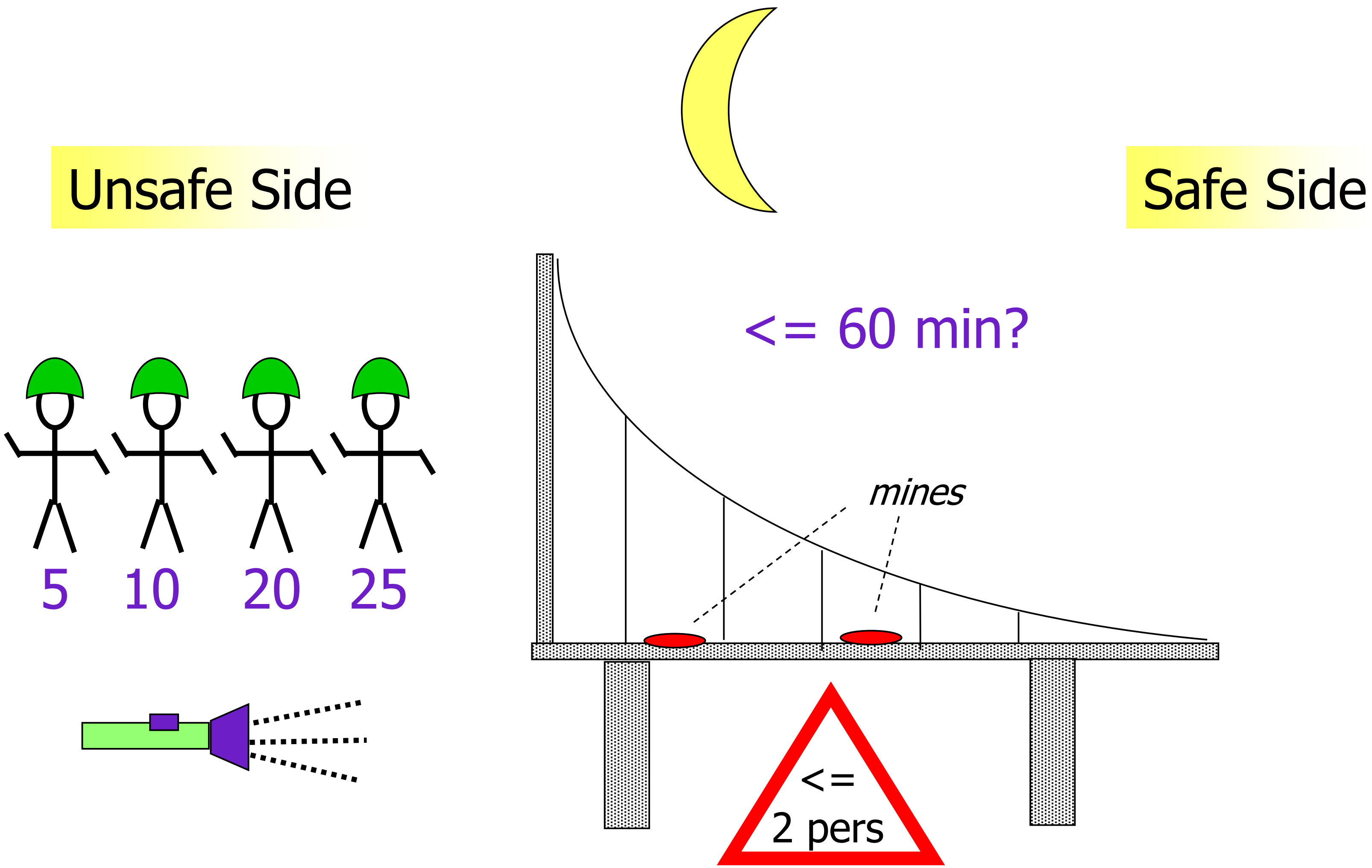
Objectives

- Review the basic constructs of Promela by carrying out a simply modeling problem

Outline

- Statement of problem
- Processes
- Basic data types and expressions
- Commands
- Communication primitives

Problem Statement



Problem Statement

Four soldiers who are heavily injured, try to flee their home land. The enemy is chasing them and in the middle of the night they arrive at a bridge that spans a river which is the border between the two countries at war. The bridge has been damaged and can only carry two soldiers at a time. Furthermore, several land-mines have been placed on the bridge and a torch is needed to sidestep all the mines. The enemy is on their trail, so the soldiers know that they have only 60 minutes to cross the bridge. The soldiers only have a single torch and they are not equally injured. The following table lists the crossing times (one-way!) for each of the soldiers:

soldier S0	**5 minutes**
soldier S1	**10 minutes**
soldier S2	**20 minutes**
soldier S3	**25 minutes**

Does a schedule exist which gets all four soldiers to the safe side within 60 minutes?

Hint: Before proceeding, it may be instructive to solve the soldiers problem on paper before proceeding.

(Ruys & Brinksma 1998)

For You To Do…

- Pause the lecture…
- Try to solve the soldiers problem on paper before proceeding.

Assessment

- Even though the problem statement suggests a strong connection to “time”, we don’t need a special notion of “clock” or a real-time model-checker to model this problem.
- In essence, this is a *scheduling/planning* problem, and we will be taking advantage of the fact that a model-checker explores all possible orders of instruction interleavings.
- Thus, we can easily have the model-checker construct all the different orderings for moving soldiers across the bridge.

Modeling: Basic Ideas

There is communication of soldiers between two components: *unsafe side* and *safe side*

Unsafe Side

Safe Side

`proctype Unsafe()`

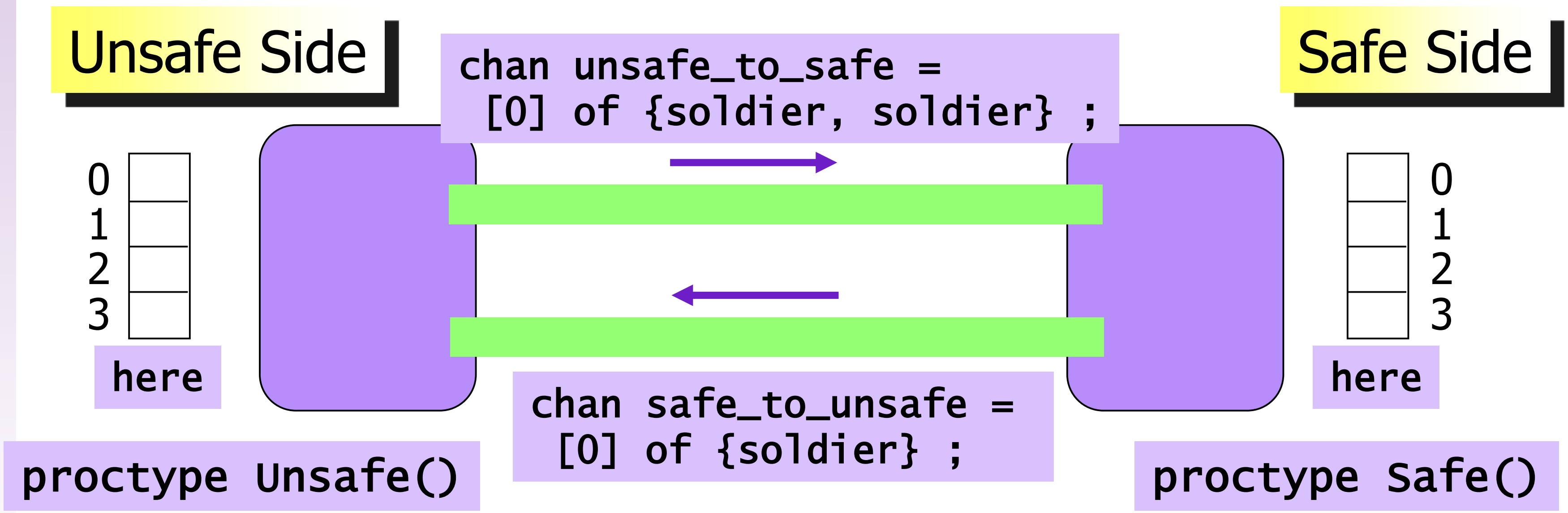
`proctype Safe()`

Modeling: Basic Ideas

There is communication of soldiers between two components: *unsafe side* and *safe side*

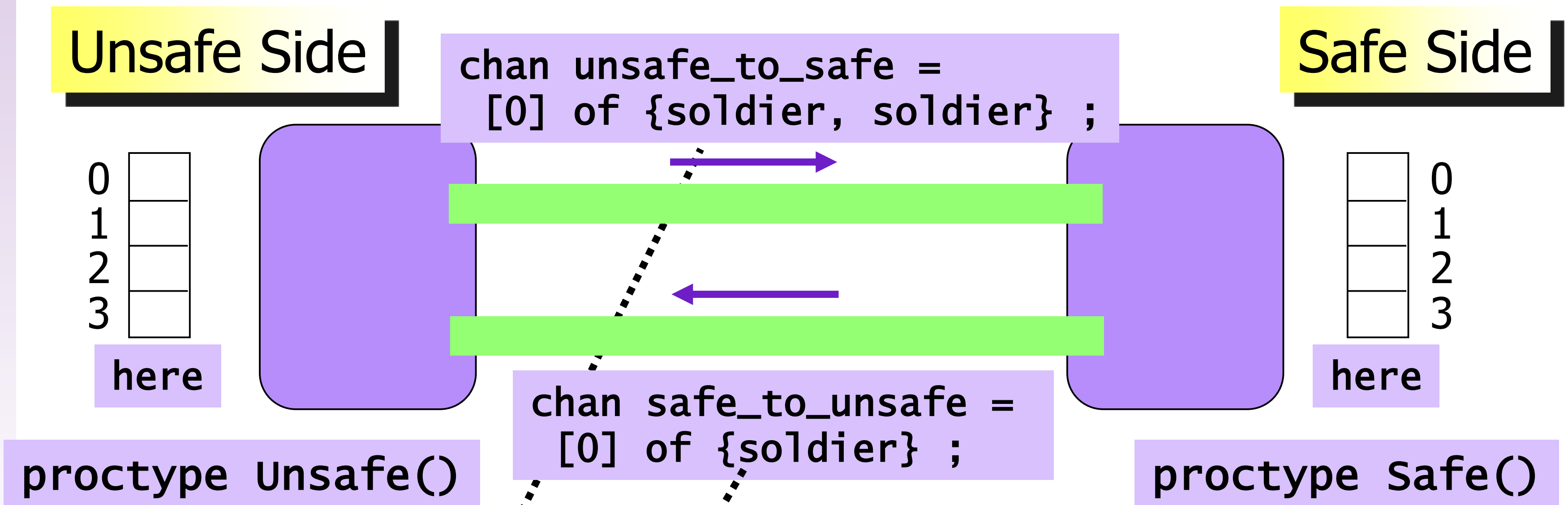
Modeling: Basic Ideas

There is communication of soldiers between two components: *unsafe side* and *safe side*



Modeling: Basic Ideas

There is communication of soldiers between two components: *unsafe side* and *safe side*



It's clear that you want to send two soldiers over but only one soldier back (this tells us what types to give to the channels above).

No need to buffer messages because you can never have more than one "package" of soldiers on bridge at a time (so channel size is 0).

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}
```

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;




  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od
}
```

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}
```

Soldiers start on unsafe side.

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;

  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od
}
```

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}
```

Non-deterministically choose two soldiers to cross, and mark them as not "here".

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;

  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od
}
```

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}
```

Send chosen soldiers across bridge.

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;

  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od
}
```

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}
```

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;

  do
  ::  unsafe_to_safe ? s1, s2 ;
                 = 1 ;
                 = 1 ;
                 ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od
}
```

Exit if all soldiers are gone.

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}
```

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;

  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od
}
```

Receive soldiers from safe side.

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}
```

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;

  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od
}
```

Mark soldiers as "here"

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}
```

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;

  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od
}
```

Advance stopwatch by time associated with slowest (highest #) soldier.

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe
      IF all_gone ->
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}
```

Exit if all soldiers here.

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;

  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od
}
```

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldie
      unsafe_to_saf
      IF all_gone -
      safe_to_unsaf
      here[s1] = 1
      stopwatch ! s
  od
}
```

Non-deterministically select from soldiers here to return to other side, and mark soldier as not here.

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;

  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od
}
```

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
      safe_to_unsafe
      here[s1] = 1 ;
      stopwatch ! s1
  od
}
```

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;

  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od
}
```

Send soldier back to unsafe side.

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}
```

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;







  do
  ::  unsafe_to_safe ? s1, s2 ;
                  = 1 ;
                  = 1 ;
                 h ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
  od
}
```

Receive soldier from unsafe side.

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}
```

Mark received soldier as "here"

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;

  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
         all_here -> break FI ;
         ect_soldier(s1) ;
         e_to_unsafe ! s1
  od
}
```

Sides as Processes

```
proctype Unsafe()
{
  bit     here[N] ;
  soldier s1, s2 ;
  here[0] = 1 ;
  here[1] = 1 ;
  here[2] = 1 ;
  here[3] = 1 ;

  do
  ::  select_soldier(s1) ;
      select_soldier(s2) ;
      unsafe_to_safe ! s1, s2 ;
      IF all_gone -> break FI ;
      safe_to_unsafe ? s1 ;
      here[s1] = 1 ;
      stopwatch ! s1 ;
  od
}
```

```
proctype Safe()
{
  bit     here[N] ;
  soldier s1, s2 ;

  do
  ::  unsafe_to_safe ? s1, s2 ;
      here[s1] = 1 ;
      here[s2] = 1 ;
      stopwatch ! max(s1, s2) ;
      IF all_here -> break FI ;
      select_soldier(s1) ;
      safe_to_unsafe ! s1
```

Advance stopwatch with time associated with received soldier.

Soldier Booking

```
#define N              4
#define soldier        byte

#define select_soldier(x) \
if                        \
:: here[0] -> x=0         \
:: here[1] -> x=1         \
:: here[2] -> x=2         \
:: here[3] -> x=3         \
fi ;                      \
here[x] = 0

#define all_gone    (!here[0] && !here[1] && !here[2] && !here[3])
#define all_here    (here[0] && here[1] && here[2] && here[3])
```

Non-deterministically select a soldier that is here and assign the soldier's # to `x`. *Then mark soldier as "not here".*

Soldier Bookkeeping

```
#define N              4
#define soldier        byte

#define select_soldier(x) \
if                        \
:: here[0] -> x=0         \
:: here[1] -> x=1         \
:: here[2] -> x=2         \
:: here[3] -> x=3         \
fi ;                      \
here[x] = 0

#define all_gone    (!here[0] && !here[1] && !here[2] && !here[3])
#define all_here    (here[0] && here[1] && here[2] && here[3])
```

Test for "all here" or "all gone"

Miscellaneous Macros

```
#define MSCTIME      printf("MSC: %d\n", time)
#define IF           if ::
#define FI           :: else fi
#define max(x,y)     ((x>y) -> x : y)
```

Used to display times in message sequence chart

Miscellaneous Macros

```
#define MSCTIME      printf("MSC: %d\n", time)
#define IF           if ::
#define FI           :: else fi
#define max(x,y)     ((x>y) -> x : y)
```

Handy for simple conditionals with no "else"

Miscellaneous Macros

```
#define MSCTIME      printf("MSC: %d\n", time)
#define IF           if ::
#define FI           :: else fi
#define max(x,y)     ((x>y) -> x : y)
```

Implement MAX function

Timer Process

```
proctype Timer()
{
end:
  do
  ::  stopwatch ? 0 -> atomic { time=time+5  ; MSCTIME }
  ::  stopwatch ? 1 -> atomic { time=time+10 ; MSCTIME }
  ::  stopwatch ? 2 -> atomic { time=time+20 ; MSCTIME }
  ::  stopwatch ? 3 -> atomic { time=time+25 ; MSCTIME }
  od
}
```

Note: the reason that we might have a separate process to increment these timing values is because we don't want to duplicate the code in both the *safe* and *unsafe* processes. SPIN does not have functions/procedures so one can only "factor out" code by (a) using a macro, (b) implementing the functionality in a separate process and passing parameters to it using a channel (e.g., `stopwatch` above).

Timer Process

```
proctype Timer()
{
end:
  do
  ::  stopwatch ? 0 -> atomic { time=time+5  ; MSCTIME }
  ::  stopwatch ? 1 -> atomic { time=time+10 ; MSCTIME }
  ::  stopwatch ? 2 -> atomic { time=time+20 ; MSCTIME }
  ::  stopwatch ? 3 -> atomic { time=time+25 ; MSCTIME }
  od
}
```

Unique ID of soldier

Increment clock by "time weight" associated with soldier.

Note: the reason that we might have a separate process to increment these timing values is because we don't want to duplicate the code in both the *safe* and *unsafe* processes. SPIN does not have functions/procedures so one can only "factor out" code by (a) using a macro, (b) implementing the functionality in a separate process and passing parameters to it using a channel (e.g., `stopwatch` above).

Timer Process

```
proctype Timer()
{
end:
  do
  ::  stopwatch ? 0 -> atomic { time=time+5  ; MSCTIME }
  ::  stopwatch ? 1 -> atomic { time=time+10 ; MSCTIME }
  ::  stopwatch ? 2 -> atomic { time=time+20 ; MSCTIME }
  ::  stopwatch ? 3 -> atomic { time=time+25 ; MSCTIME }
  od
}
```

"end" label signifies that the state at the beginning of the do-loop is a valid end-state.

Note: the reason that we might have a separate process to increment these timing values is because we don't want to duplicate the code in both the *safe* and *unsafe* processes. SPIN does not have functions/procedures so one can only "factor out" code by (a) using a macro, (b) implementing the functionality in a separate process and passing parameters to it using a channel (e.g., `stopwatch` above).

For You To Do…

- Pause the lecture…
- Run SPIN in simulation mode to step through a complete schedule of soldiers crossing the bridge.
- Now let’s consider an exhaustive verification with SPIN -- how can you get SPIN to search and display a schedule that allows the soldiers to move to the safe side in 60 minutes or less?

Finding A Solution

- We would like to get SPIN to generate a trace for us that illustrates a schedule where the soldiers can get from the unsafe to the safe side in 60 minutes or less.
- SPIN generates *counter-example* or *property-violation* traces. Therefore, we need to ask SPIN to try to prove that a schedule *does not exist*. If it finds a counterexample, it will illustrate a schedule that's a solution to the soldiers problem.
- Using Linear Temporal Logic, we can formalize the desired property directly (as illustrated on the next slide).
- One can also get SPIN to generate an appropriate trace using an assertion.

Temporal Property

Eventually, the variable `time` has a value greater than 60.

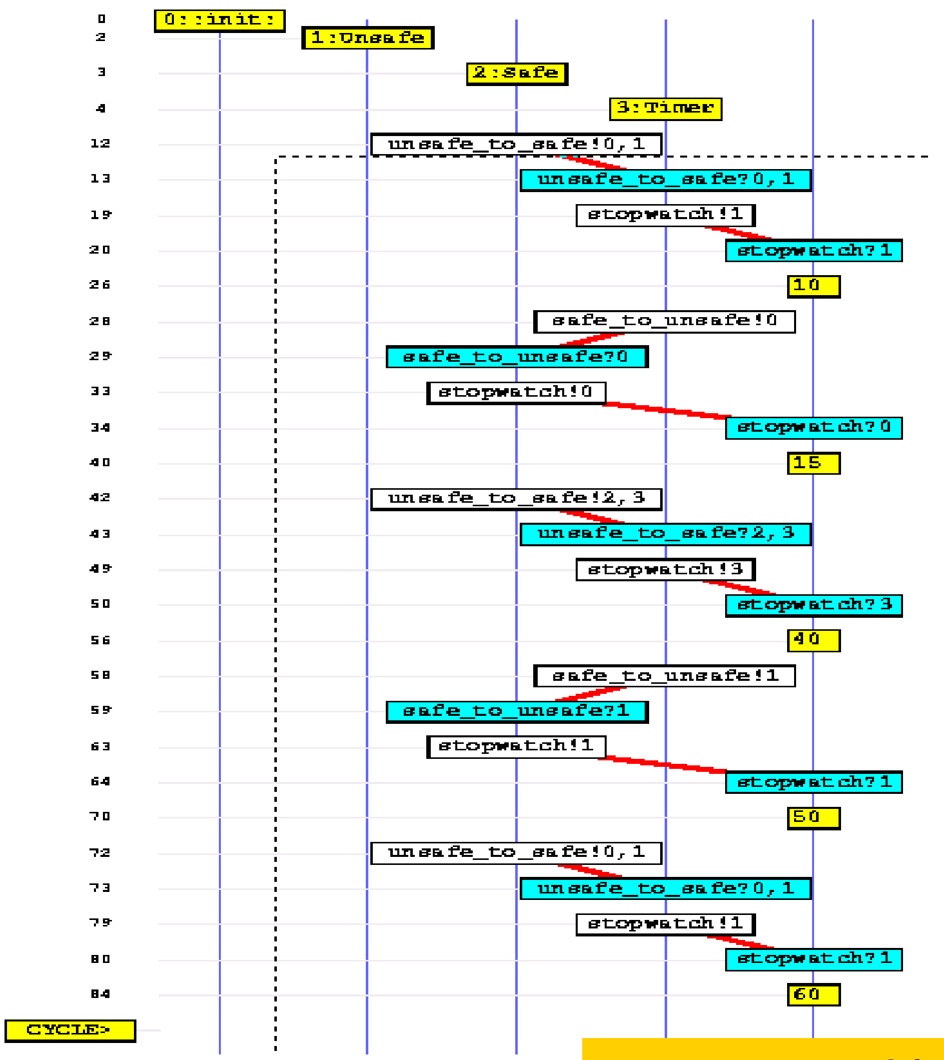
...is expressed in Linear Temporal Logic as...

```
<>(time > 60)
```

SPIN Steps

- Define the predicate time > 60 in `soldiers.prom` file
  - `#define outoftime   time > 60`
- Put the negation of the desired LTL property in `soldiers.ltl`
  - `(!<>(outoftime))`
- Run SPIN to create a verifier based on the property
  - `spin -a -F soldiers.ltl soldiers.prom`
- Compile
  - `gcc -o pan.exe pan.c`
- Run with command-line option (-a) specifying that a liveness property is being checked
  - `pan.exe -a`
- Display error trail as message sequence chart (creates `soldiers.ps`)
  - `spin -t -M soldiers.prom`

Message Sequence Chart



Soldiers 0,1 → Safe

Total time: 10min

Soldiers 0 → Unsafe

Total time: 15min

Soldiers 2,3 → Safe

Total time: 40min

Soldiers 1 → Unsafe

Total time: 50min

Soldiers 0,1 → Safe

Total time: 60min

spin –t –M soldiers.prom

For You To Do…

- Pause the lecture…
- Download the file soldiers.prom from the web page and carry out the steps described on the previous two slides.
- An appropriate schedule can also be generated using an assertion, but you will probably have to tweak the model in `soldiers.prom` slightly. Generate an appropriate schedule using an assertion.

Acknowledgements

- The Soldiers Problem description and solution is taken from a paper by Ruys and Brinksma at the TACAS 1998 (Lecture Notes in Computer Science 1384).
- Thanks to Theo Ruys for the Promela source code.