First-class synchronous operations

```
type 'a event

val sendEvt : ('a chan * 'a) -> unit event
val recvEvt : 'a chan -> 'a event

val wrap    : ('a event * ('a  -> 'b)) -> 'b event
val guard   : (unit -> a event) -> a event
val choose  : 'a event list -> 'a event

val sync    : 'a event -> 'a
val select  : 'a event list -> 'a

val never     : 'a event
val alwaysEvt : 'a -> 'a event
```

Additional synchronous operations: I-variables

```
structure SyncVar : sig
    type 'a ivar
    val iVar : unit -> 'a ivar
    val iPut : ('a ivar * 'a) -> unit
    val iGet : 'a ivar -> 'a
    val iGetEvt : 'a ivar -> 'a event
    ...
  end = ...
```

Additional synchronous operations: Mailboxes

```
structure Mailbox : sig
    type 'a mbox
    val mailbox : unit -> 'a mbox
    val send : ('a mbox * 'a) -> unit
    val recv : 'a mbox -> 'a
    val recvEvt : 'a mbox -> 'a event
    ...
  end = ...
```

Futures in CML

```
val future : ('a -> 'b) -> 'a -> 'b event
```

```
structure SV = SynchVar

fun future f x = let
      val cell = SV.iVar()
      in
        spawn (fn () => SV.iPut(cell, f x));
        SV.iGetEvt cell
      end
```

Futures in CML --- exceptions

```
datatype 'a result = RES of 'a | EXN of exn

fun future f x = let
      val cell = SV.iVar()
      fun f' x = RES(f x) handle ex => EXN ex
      in
        spawn (fn () => SV.iPut(cell, f' x));
        wrap (
          SV.iGetEvt cell,
          fn (RES x) => x | EXN ex => raise ex)
      end
```

Promises in CML

Promises are asynchronous remote procedure calls.

```
structure Remote : sig
  type request
  type reply
  val request : (request * reply SyncVar.iVar)
        -> unit
  val flush : unit -> unit
end = ...
```

```
val promise : Remote.request -> Remote.reply event
```

Promises in CML

```
fun promise (x : Remote.request) = let
      val replV = SV.iVar()
      in
        Remote.request (x, replV);
        SV.iGetEvt replV
      end
```

Promises in CML

```
fun promise (x : request) = let
      val replV = SV.iVar()
      in
        Remote.request (x, replV);
        guard (fn () => (
          case SV.iGetPoll replV
           of NONE => (
                Remote.flush();
                SV.iGetEvt replV)
            | SOME repl => alwaysEvt repl
      end
```