Lecture 22

Note Title 4/10/2008

Recall synch comm primitives

channels
send Evts
recv Evts
sync (single event)
select (list of events)

} potential for communication

} Project Concurrent ML CML

CML also has event combinators — functions that produce new events from existing events

wrap (e, f) when synchronized upon does the communication denoted by e and calls f on the result.

choose [$e_1, e_2, \ldots e_n$] is an event that synchronizes one of $e_1, \ldots e_n$

select [$e_1, \ldots e_n$] $\equiv$ sync (choose [$e_1, \ldots e_n$])

other combinators: (ordinary value)

(alwaysEvt a) ⟶ evt that yields a when synchronized

neverEvt → never successfully synchronizes

(timeoutEvt t)

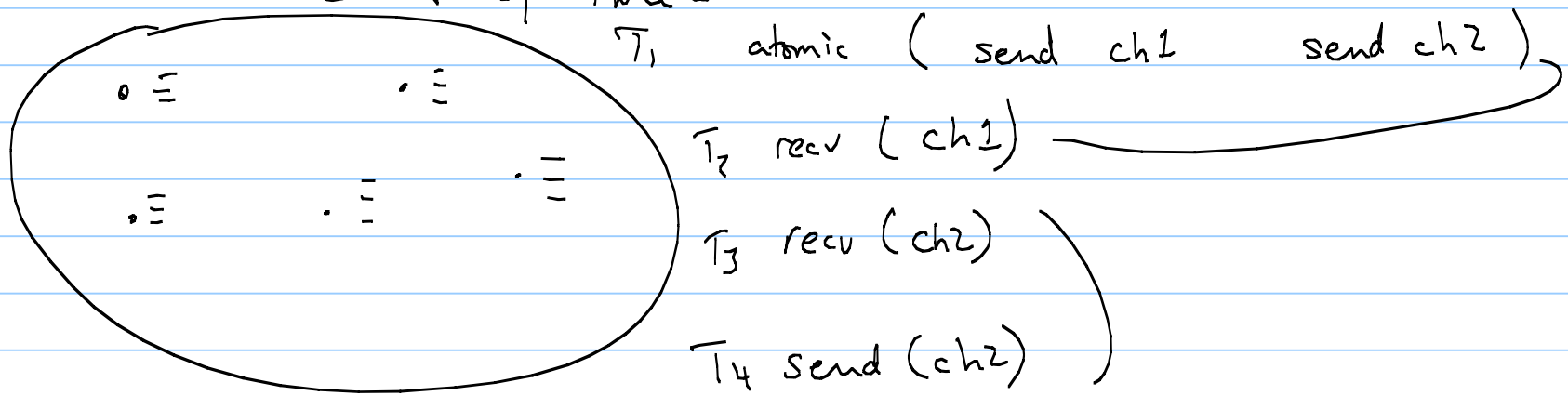
Fairly sophisticated comm. patterns can be built in event values — but not all interesting ones.

Consider

guarded receive — look at the value to be received and only take it if it is acceptable.

grecvEvt g ch : (g: boolean-valued function) only receives value x from ch if (g x) is true.

Transactional Events – Fluet & Donnelly ICFP 2006 ~10p.
" " Int'l Journal of Functional Programming
2007-2008 — 79pp



Power of The primitives: 1) emulate CML (not too surprising)
2) emulate STM

Implementation: currently based on STM

End of Synch Communication

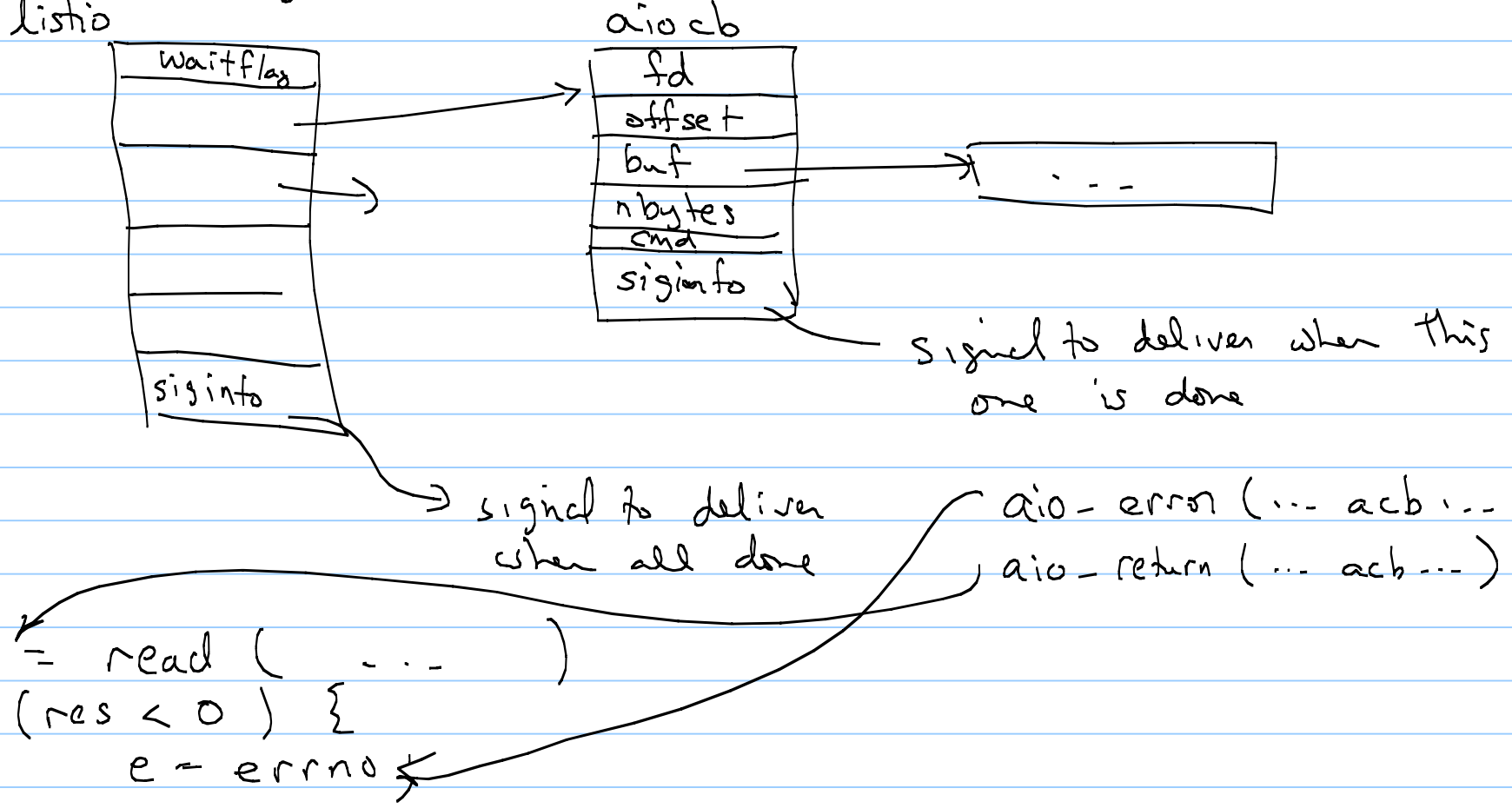
locks, monitors, cv's, shared memory.

Asynchronous concurrent programming — doing concurrent i/o from a single thread — POSIX Asynch I/O interface, Win32 interface
- contrast to read, write syscalls.

- idea: prog. can launch multiple I/O ops; later find out the result by receiving an interrupt (signal) or by polling
- poll/select system call model — ask of a set of fds "are you ready to do i/o" then when one is, you issue the read or write

Posix.4:

Source: Programming for the Real World, Bill O. Gallmeister. O'Reilly.



Signal handling: — C fuction set up as handler for a specific signal number.
passed the address of aiocb assoc. w/ completed operation

```
if aio_error(acb) == EINPROGRESS { return }

if aio_return(acb) != acb->aio_nbytes {
        maybe a problem

}
```

synchronization — protecting against concurrent access by a signal handler.

```
sigprocmask(SIG_BLOCK, &completionSignals, &prevMask)
... do critical section ...
sigprocmask(SIG_SETMASK, &prevMask, NULL)
```

Why use AIO?

Use multiple resources concurrently

- other ways to accomplish this
  - threads (but sometimes implemented w/ AIO)
  - poll/select (also used to implemet threads)

aio: start some i/o
find out when done
start some i/o

signals to identify being done (painful)

poll/select:
wait for ready
do some i/o
wait for ready } compute for awhile then wait for ready.
:

to program nicer but not quite as responsive