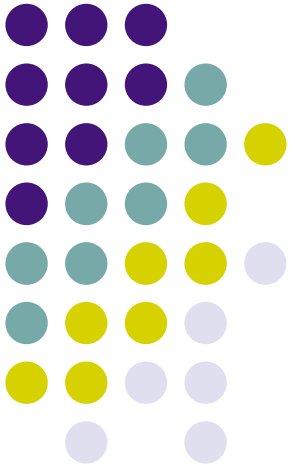
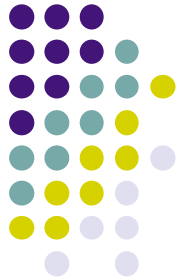
Message Sending

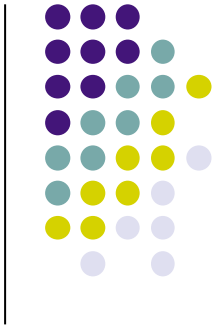


Message Sending: Properties



- Message sending
  - asynchronous
  - ordered per thread
  - no order from multiple threads
  - first-class messages

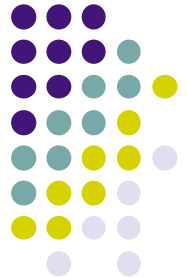
Asynchronous Sending



```
P={NewPort S}
thread … {Send P M} … end          (1)
thread … {Process S} … end         (2)
```

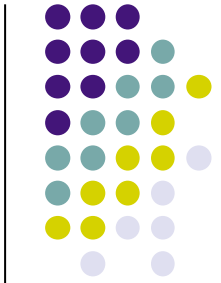
- Asynchronous: (1) continues immediately after sending
- Sender does not know when message processed
  - message processed eventually

Asynchronous Reply



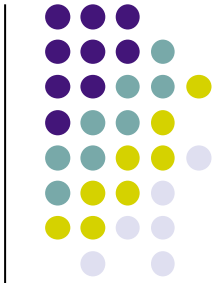
- Sender sends message containing dataflow variable for answer
  - does not wait for receipt
  - does not wait for answer when sending
- Waiting for answer, only if answer needed
- Helps to avoid latency
  - sender continues computation
  - receiver might already deliver message

Synchronous Sending



- Sometimes more synchronization needed
  - sender wants to synchronize with receiver upon receipt of message
  - known as: ***handshake, rendezvous***
- Can also be used for delivering reply
  - sender does not wait for reply computed, or
  - sender waits until reply computed

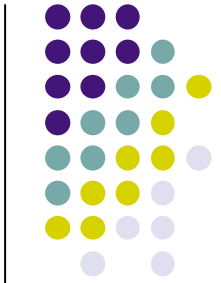
Waiting for Variables



- How to express that execution resumes only if variable `X` bound?
- Notice that conditional is suspendable

```
proc {Wait X}
   if X==1 then skip else skip end
end
```

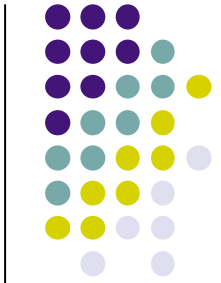
Synchronous Send



```
proc {SyncSend P M}
   Ack in {Send P M#Ack}
   {Wait Ack}
end

proc {Process MA}
   case MA of M#Ack then
      Ack=okay …
   end
end
```

Asynchronous Send

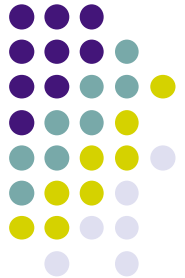


- Synchronous send can be turned into asynchronous send again by use of threads

```
proc {AsyncSyncSend P M}
   thread {SyncSend P M} end
end
```

- Sending: variants can be mutually expressed

Message Order



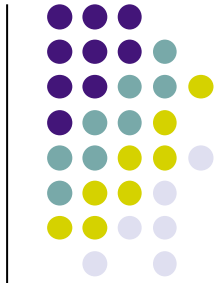
- Order on same thread: A always before B

```
thread
    … {Send P A} … {Send P B} …
end
```

- No order among threads

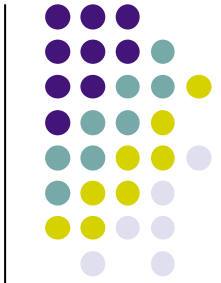
```
thread … {Send P A} … end
thread … {Send P B} … end
```

Messages



- Important aspect of agents
  - messages are first-class values: can be computed, tested, manipulated, stored
  - can contain any data structure including procedure values
- First-class messages are expressive
  - messages received stored in a log
  - agent forwards by adding time-stamp to message

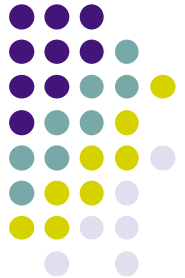
A Compute Server



```
proc {ComputeAgent M}
   case M
   of run(P) then {P}
   [] run(F R) then R={F}
   end
end
```

- Runs as an agent in its own thread
- Executes procedures contained in messages

Distribution



- Spawn computations across several computers connected by network
- Message sending important way to structure distributed programs
- Compute servers make sense in this setting
- Oz: transparent distribution