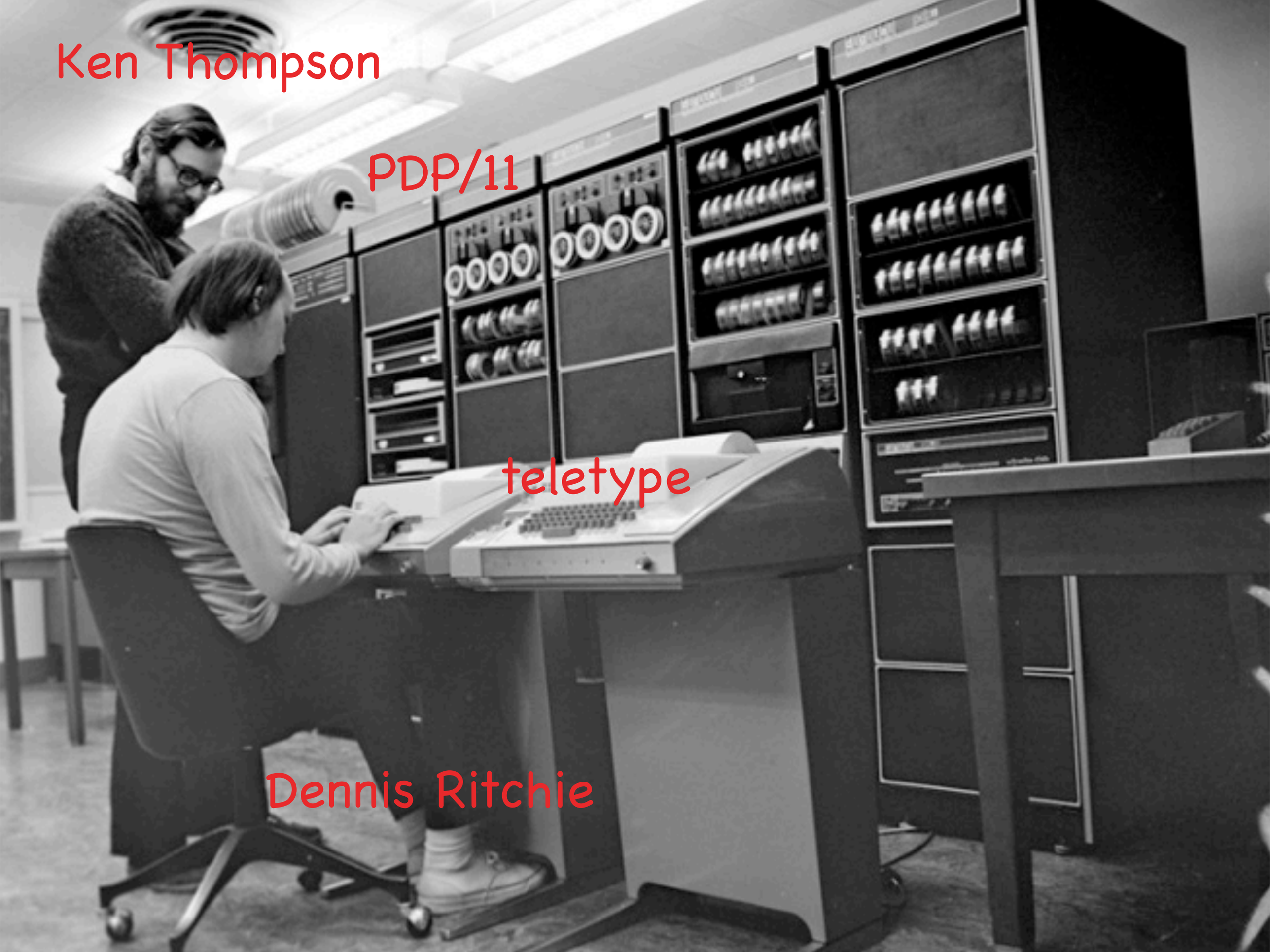
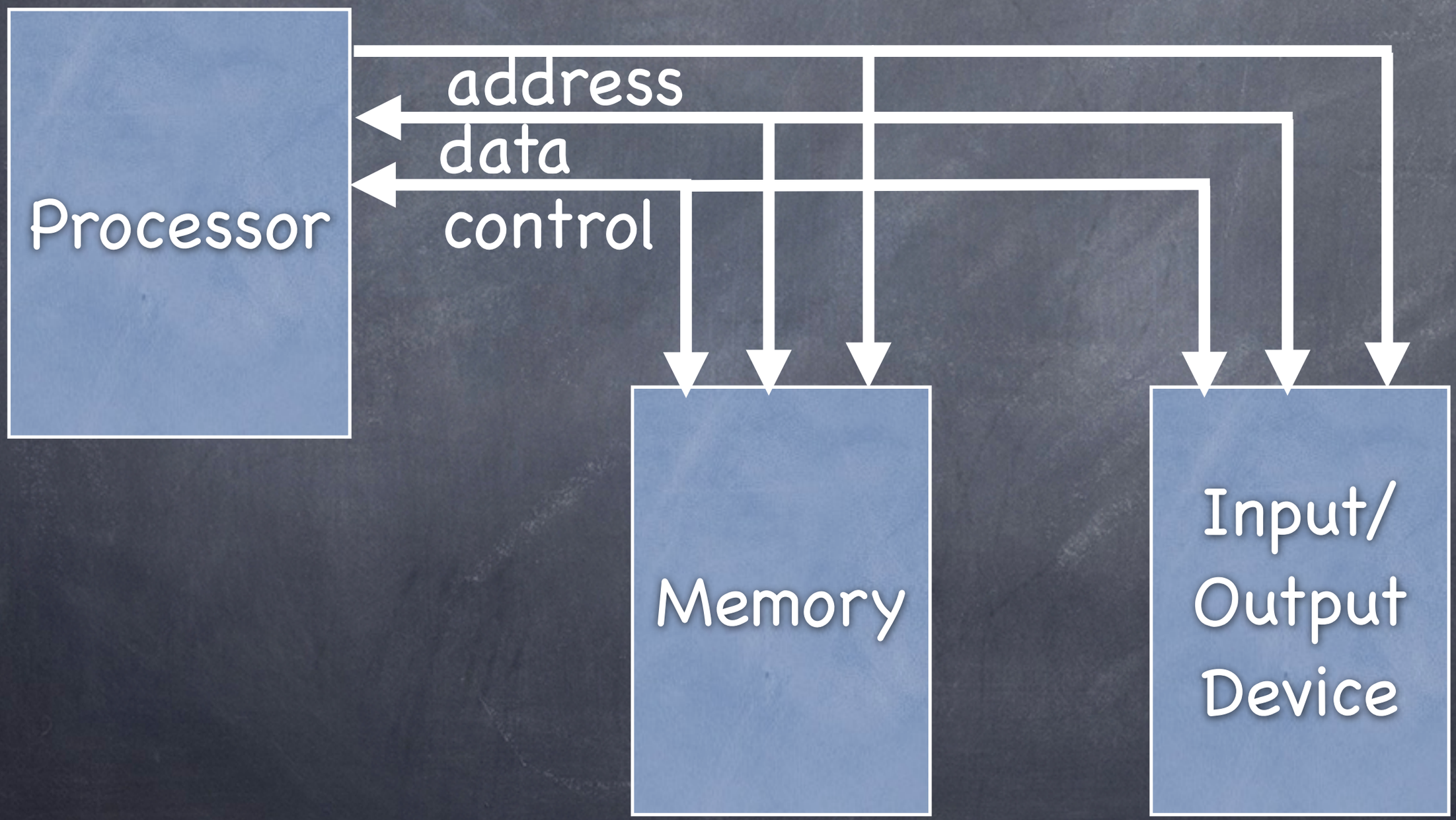
Announcements

???


Ken Thompson
PDP/11
teletype
Dennis Ritchie

Memory Mapped Input/ output



Receiver Status Register:

RCVR DONE

Receiver Data Buffer:

Received Data Bits

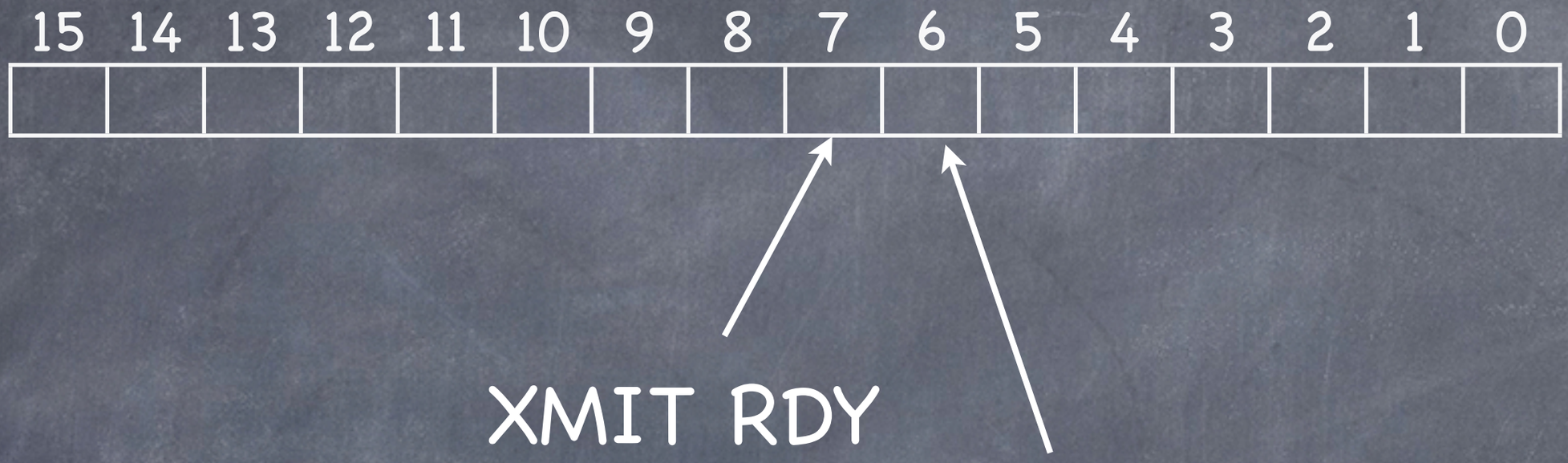

```
short *RSR = (short *) 0777564;
short *RDB = (short *) 0777566;

#define RCRVDONE 0x0080
```

	ADDRESS
LINE CLOCK	777546
	777560
CONSOLE	777662
	777564
	777566
	776XX0

```
int readLine( char * buffer, int len ) {
  int pos = 0;
  do {
    while (! (*RSR & RCVRDONE )) {
      // wait for a character
    }
    buffer[pos] = (char) (*RDB & 0x00FF);
    pos++;
  } while ( pos < len && buffer[ pos-1 ] != '\n' )
  return pos;
}
```

Transmitter Status Register:

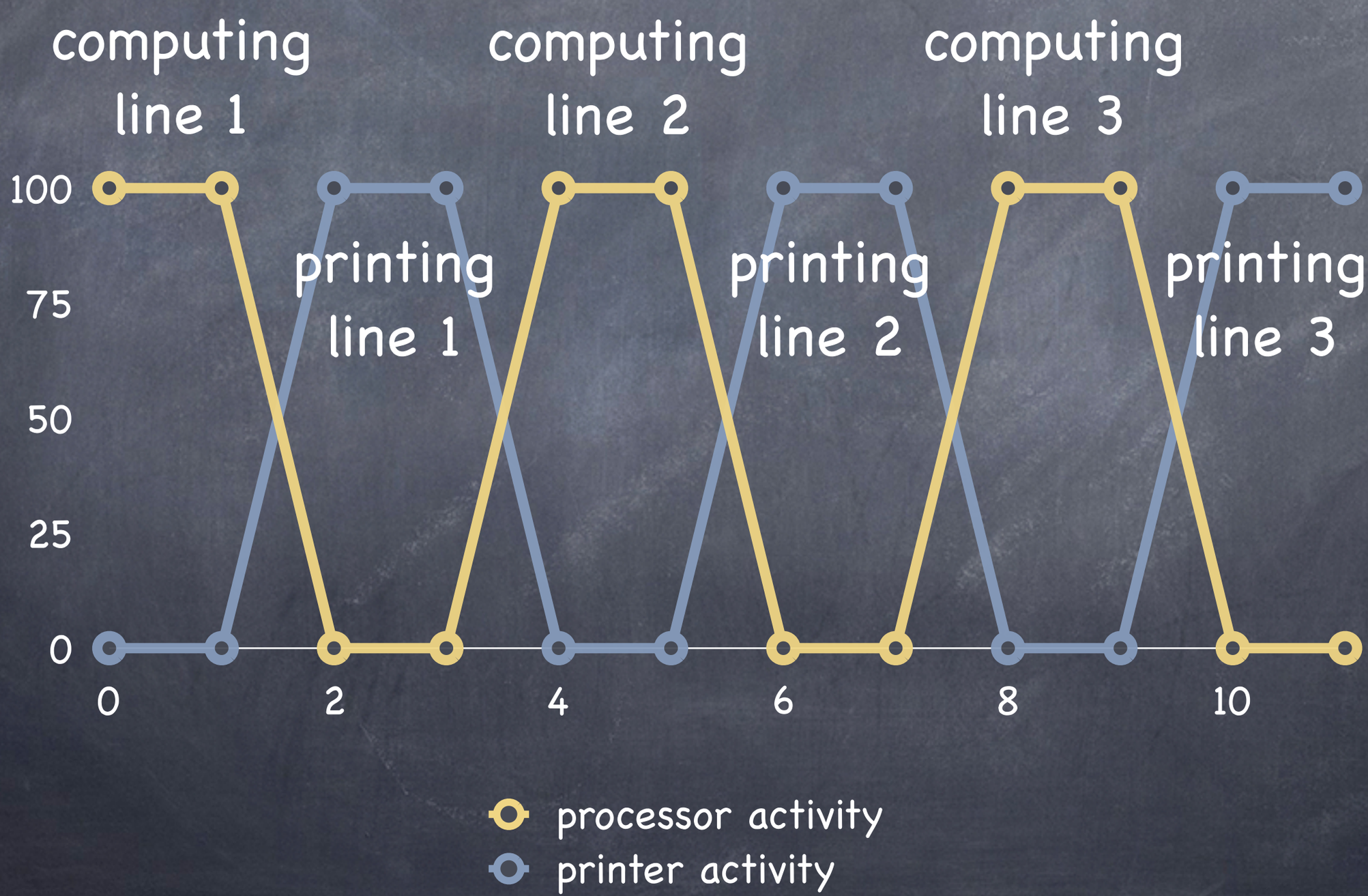


Transmitter Data Buffer:

```
void putLine( char * buffer ) {
  int pos = 0;
  while ( buffer[ pos ] != 0 ) {
    while ( ! (*TSR & XMITRDY ) {
      // wait until printer ready
      }
      *TDB = buffer[pos];
      pos++
    }
}
```

```
int main( int argc, char * argv[] ) {
  ...
  int i
  while ( happy ) {
    i++;
    // compute contents of line[i]
       . . .
    putLine( line[i] );
    }
}
```

I/O Delays


computing line 1
computing line 2
computing line 3
printing line 1
printing line 2
printing line 3
100
75
50
25
0
0
2
4
6
8
10
processor activity
printer activity

Exception Handling Approaches

- Branch to "fixed" address
  - Actually fixed
  - Address stored in fixed address (typically low or high addresses in memory).
  - Address stored in special register
- Save information including PC where error occurred (like bal saves return address).
- Enter supervisor mode!

I/O Interrupt Handling Approaches

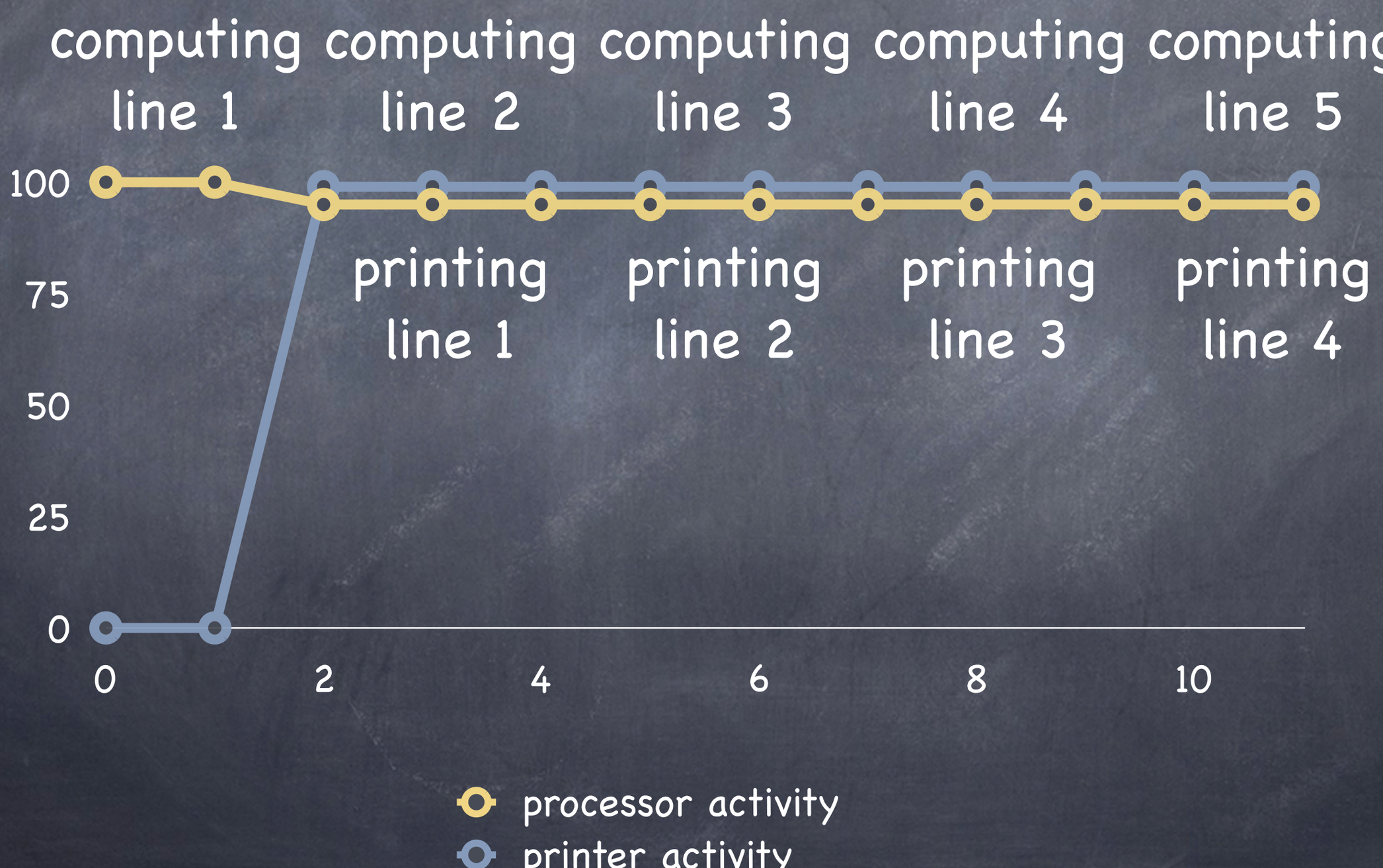
- Branch to "fixed" address
  - Actually fixed
  - Address stored in fixed address (typically low or high addresses in memory).
  - Address stored in special register
- Save information including PC where error occurred (like bal saves return address).
- Enter supervisor mode!

```
void putLine( char * buffer ) {
  for ( p = 0; p < buffer.length; p++ ) {
    add( printqueue, buffer[ pos ] );
    }
  if ( *TSR & XMITRDY ) {
     startPrint();
     }
}
```

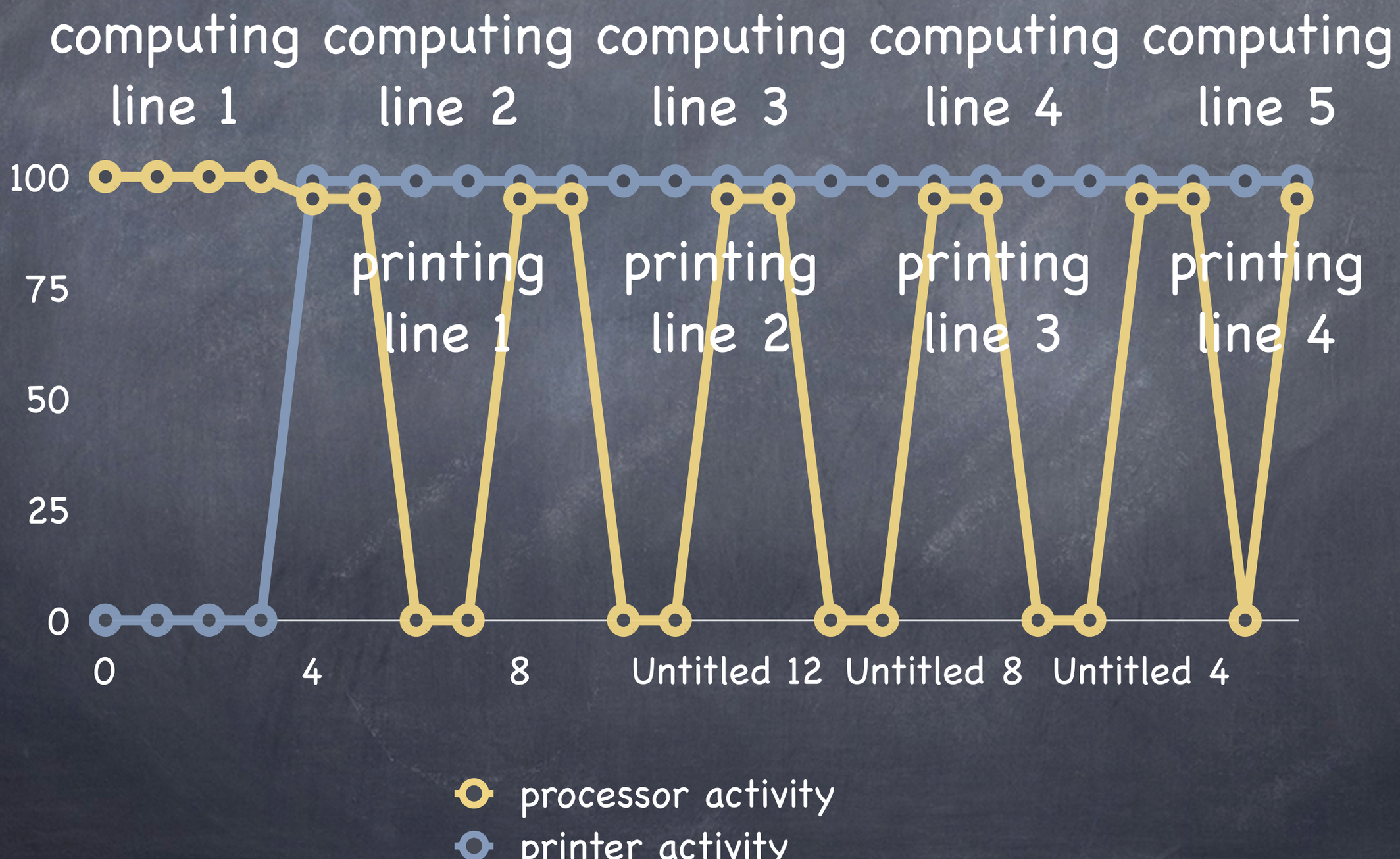
```
void startPrint() {
    ch = remove( printQueue );
    *TDB = ch;
}
```

```
void handlePrinterInterrupt( ) {
  if ( ! empty( printQueue) &&
       *TSR & XMITRDY ) {
     startPrint();
   }
```

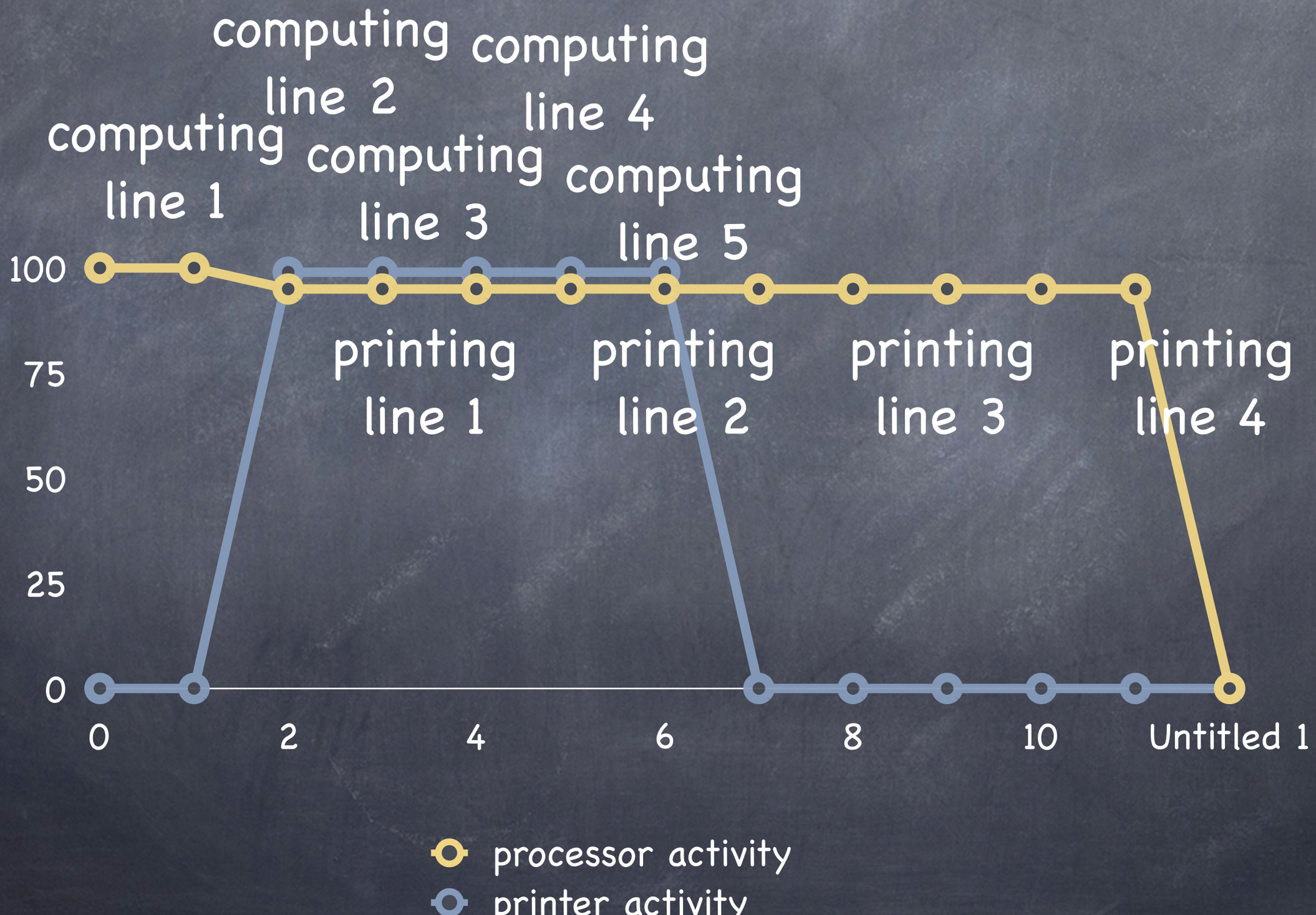
I/O / Processor Overlap



I/O / Processor Overlap



I/O / Processor Overlap



Input

```
printf( "Enter a word:\n" );
scanf("%s", input );
printf( "You typed %s\n", input );
```

Input

```
for ( x = 0; x < 2000; x++ ) {
  for ( y = 0; y < 1000000; y++ ) {
    sum = x + y + sum;
  }
}

printf( "Enter a word:\n" );
scanf("%s", input );
printf( "You typed %s\n", input );
```

```
void handleKeyboardInterrupt( ) {
    add( inputQueue, (char) (*RDB & 0x00FF) );
}
```

```
void * getLine( char buffer[] ) {
    int pos = 0;
    do {
      buffer[ pos ] = blockingRemove( inputQueue );
    } while ( buffer[pos-1] != '\n' )
    buffer[ pos ] = 0;
}
```

```
char blockingRemove( Q inputQueue) {
    while( empty( inputQueue ) ) {
      // wait for more input ...
    }
    return remove( inputQueue );
}
```

```
void * getLine( char buffer[] ) {
    int pos = 0;
    do {
      buffer[ pos ] = blockingRemove( inputQueue );
    } while ( buffer[pos-1] != '\n' )
    buffer[ pos ] = 0;
}
```

```
char blockingRemove( Q inputQueue) {
    while( empty( inputQueue ) ) {
      yieldProcessor(self);
    }
    return remove( inputQueue );
}
```

PCB
(Process Control Block)

- Collection of values describing state of process (i.e., executing program)

```
struct/class PCB {
  int savedPC;
  int savedRegs[];
  pageTableEntry pageTable[];
  ...

}
```

Ready List

- Set of PCBs of programs eager to use processor

```
void yieldProcessor( current ) {
    readyList = readyList - current;
    current = any element of readyList;
    processorState = stateOf( current );
}
```

```
void handleKeyboardInterrupt( ) {
    add( inputQueue, (char) (*RDB & 0x00FF) );
    if ( some program is awaiting input ) {
      readyList = readyList + awaiting program
    }
}
```

```
char blockingRemove( Q inputQueue) {
    while( empty( inputQueue ) ) {
      await( inputQueue );
      yieldProcessor(self);
    }
    return remove( inputQueue );
}
```

Timer Interrupts

	ADDRESS	VECTOR	PRIORITY
LINE CLOCK	777546 777560	100	BR6
CONSOLE	777662 777564 777566 776XX0	60/64	BR4

Time Slicing

```
void timerInterrupt( ) {
    stateOf( current ) = processorState;
    current = any element of readyList;
    nextTimerInterrupt = now + slice;
    processorState = stateOf( current );
}
```