



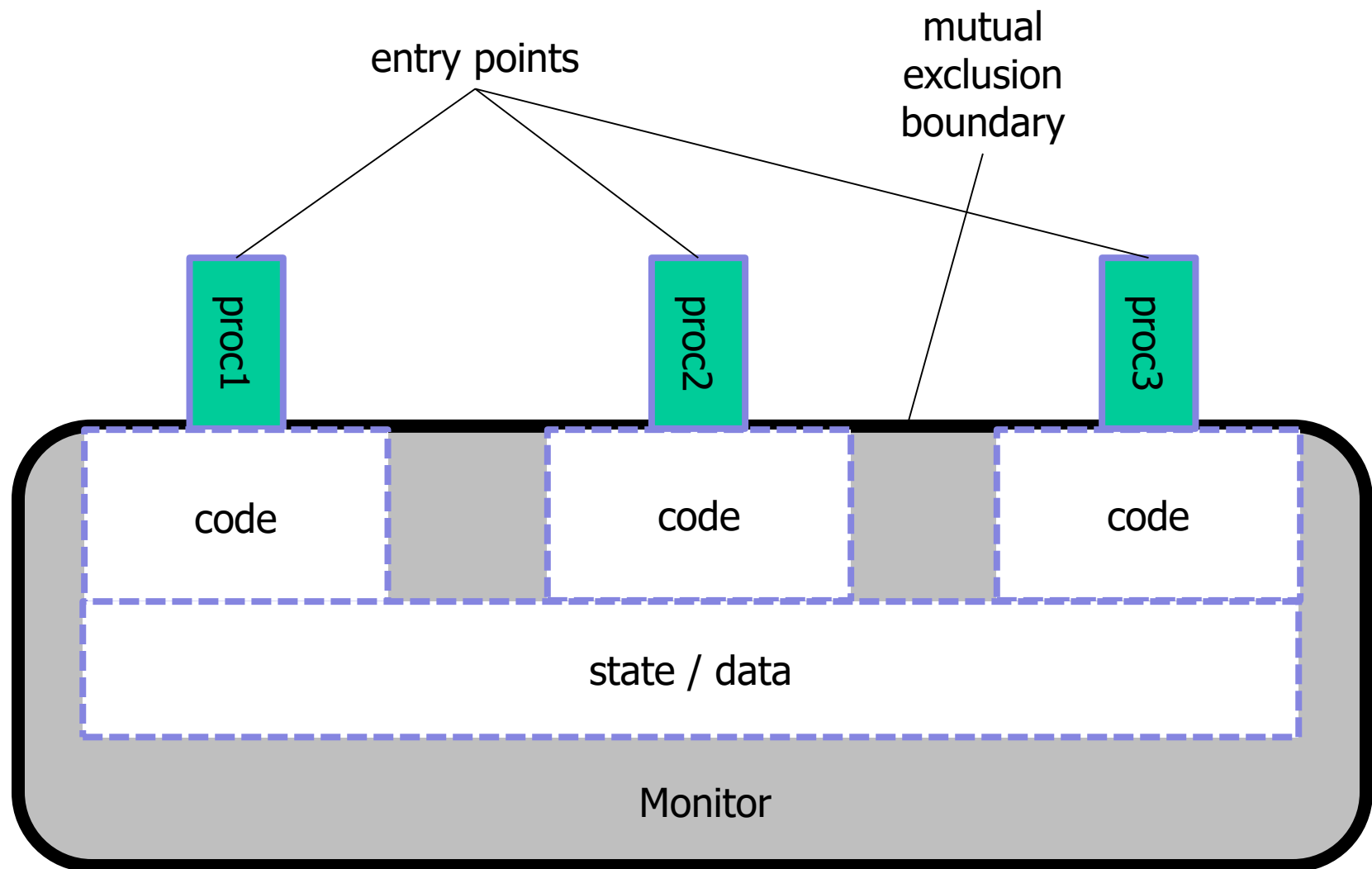
Ταυτόχρονος Προγραμματισμός (ECE321)

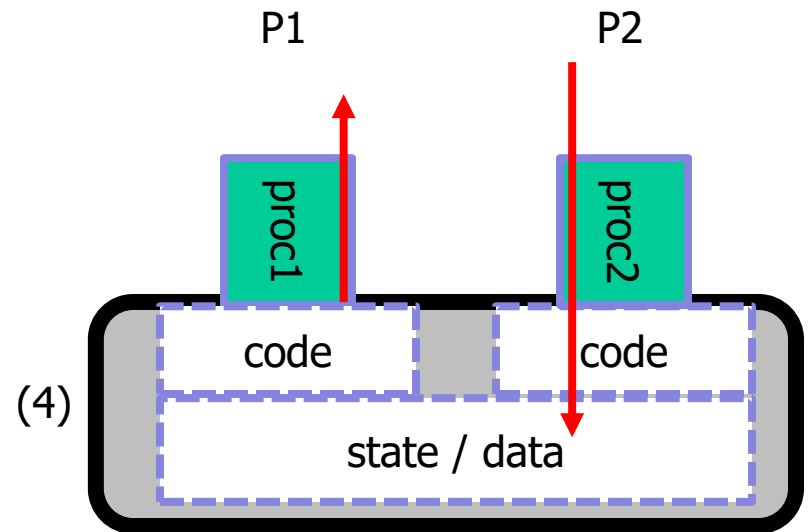
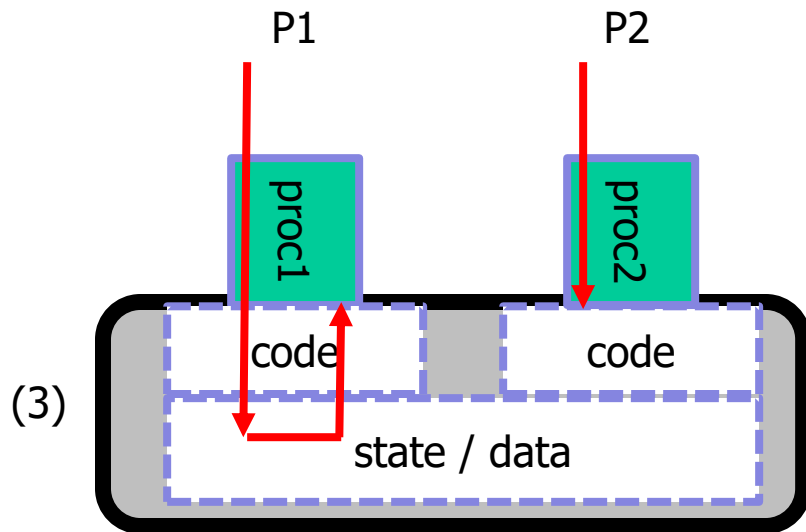
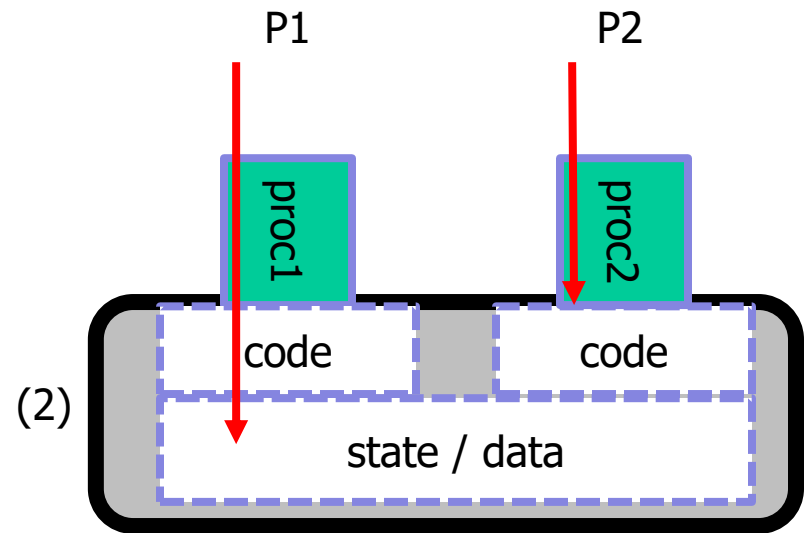
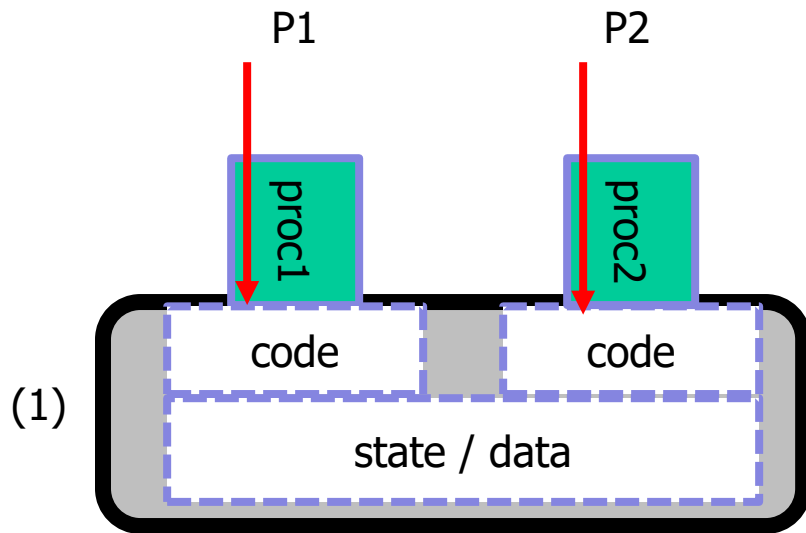
#10

Ελεγκτές/Παρατηρητές
(monitors)

Ελεγκτής

- Λειτουργίες μέσα σε ένα συγκεκριμένο «πλαίσιο» συγχρονισμού με **αυτόματο** αμοιβαίο αποκλεισμό
 - βολικός μηχανισμός για ένα **τμήμα λογισμικού** που προσπελάζεται με **δομημένο** τρόπο, μέσω ενός API
- Ο συγχρονισμός γίνεται την στιγμή που ένα νήμα **επιχειρεί** να καλέσει μια λειτουργία του ελεγκτή
- Αν κάποιο άλλο νήμα **ήδη** εκτελεί μια λειτουργία μέσα στον ελεγκτή, το καλών νήμα **μπλοκάρεται**
 - μέχρι το νήμα που ήδη εκτελεί κώδικα μέσα στον ελεγκτή να ολοκληρώσει την εκτέλεση του και να βγει από τον ελεγκτή





Δομή κώδικα

```
monitor name {  
  
    /* δηλώσεις / αρχικοποίηση μεταβλητών */  
    ...  
  
    /* ρουτίνες πρόσβασης ελεγκτή */  
    proc1 (...) {...}  
    proc2 (...) {...}  
    ...  
    procN (...) {...}  
  
}
```

εκτελούνται με **εγγυημένο**
τον αμοιβαίο αποκλεισμό

Ασφαλής μετρητής

με δυαδικό σηματοφόρο

```
int cnt;
bsem mtx;

void init() {
    cnt = 0;
    init(mtx,1);
}

void inc() {
    down(mtx);
    cnt++;
    up(mtx);
}

void dec() {
    down(mtx);
    cnt--;
    up(mtx);
}
```

με ελεγκτή

```
monitor counter {

    int cnt;

    void init() {
        cnt = 0;
    }

    void inc() {
        cnt++;
    }

    void dec() {
        cnt--;
    }

}
```

Μεταβλητές συνθήκης

- Συχνά απαιτείται **αναμονή** υπό συνθήκη **μέσα** σε ένα κρίσιμο τμήμα
 - π.χ., αναμονή καταναλωτή όταν/όσο η αποθήκη είναι άδεια
- Η αναμονή μέσα σε ελεγκτή επιτυγχάνεται με **μεταβλητές συνθήκης** (condition variables)
 - μπορεί να θεωρηθεί ως μια **δίκαιη** ουρά αναμονής
- Οι μεταβλητές συνθήκης μπορεί να δηλωθούν μόνο **μέσα** σε έναν ελεγκτή
 - **δεν** είναι σηματοφόροι

Χρήση μεταβλητών συνθήκης

- `cond x`: δήλωση
- `wait(x)`: το καλών νήμα **μπλοκάρεται**
 - ταυτόχρονα **αίρεται** ο «συμβατικός» αμοιβαίος αποκλεισμός που ισχύει στον ελεγκτή – γιατί;
- `signal(x)`: **αφυπνίζεται** ένα (το πρώτο) από τα νήματα που έχουν μπλοκάρει στο `x`
 - αν **δεν** υπάρχει ήδη μπλοκαρισμένο νήμα στο `x`, δεν γίνεται τίποτα – η κλήση είναι άσκοπη, δεν έχει κανένα αποτέλεσμα
 - οι μεταβλητές συνθήκης **δεν** είναι σηματοφόροι
 - οι άσκοπες κλήσεις της `signal()` **δεν** είναι το ίδιο με τις χαμένες κλήσεις της `up()` σε δυαδικούς σηματοφόρους

Λύση ενός σημαντικού προβλήματος (που είχαμε με τους σηματοφόρους)

- ΚΤ που προστατεύεται από τον σηματοφόρο mt_x
- Ένα νήμα επιθυμεί να περιμένει μέσα στο ΚΤ, μπλοκάροντας σε έναν άλλο σηματοφόρο q
- Για να μην προκύψει αδιέξοδος, πρέπει πρώτα να «ελευθερωθεί» το ΚΤ, καλώντας $up(mt_x)$, **προτού** γίνει η κλήση $down(q)$
- Υπάρχει πιθανότητα **συνθήκης ανταγωνισμού**
 - μια εναλλαγή μετά το $up(mt_x)$ και πριν το $down(q)$ μπορεί να οδηγήσει σε προβλήματα στην συνέχεια
- Αυτό το πρόβλημα **δεν** υφίσταται σε ελεγκτή
 - **όμως** βλέπε ζητήματα προτεραιότητας (στην συνέχεια)

```


...
bsem mtx, q;

void init() {
    ...
    init(mtx, 1);
    init(q, 0);
}

void proc1(...) {
    down(mtx);
    ...
    if (...) {... up(mtx); }
    else {... up(mtx); down(q); }
}

void proc2(...) {
    down(mtx);
    ...
    if (...) { up(q); }
    up(mtx);
}

```



πιθανή συνθήκη
ανταγωνισμού

```

monitor {
    ...
    cond q;

    void init() {
        ...
    }

    void proc1(...) {
        ...
        if (...) { wait(q); }
    }

    void proc2(...) {
        ...
        if (...) { signal(q); }
    }
}

```

δεν υπάρχει συνθήκη ανταγωνισμού

Προτεραιότητες (1)

- Τι γίνεται αν ένα νήμα **αφυπνίσει** μέσω `signal` ένα άλλο νήμα που έχει μπλοκάρει σε μια μεταβλητή συνθήκης **μέσα** στον ελεγκτή;
 - για να ισχύει αμοιβαίος αποκλεισμός, **μόνο ένα** νήμα μπορεί να συνεχίσει την εκτέλεση του στον ελεγκτή
- **signal-continue:** προτεραιότητα έχει το νήμα που προκάλεσε την αφύπνιση (κάλεσε την `signal`)
- **signal-block:** προτεραιότητα έχει το νήμα που αφυπνίστηκε (κάλεσε `wait` προτού κληθεί η `signal`)
 - προτεραιότητα με βάση την σειρά εισόδου στον ελεγκτή
- Δεν τίθεται θέμα προτεραιότητας αν η `signal` καλείται ως τελευταία εντολή στον ελεγκτή – γιατί;

πλαίσιο συγχρονισμού
του ελεγκτή

ουρά αναμονής της
μεταβλητής συνθήκης x

cond x queue

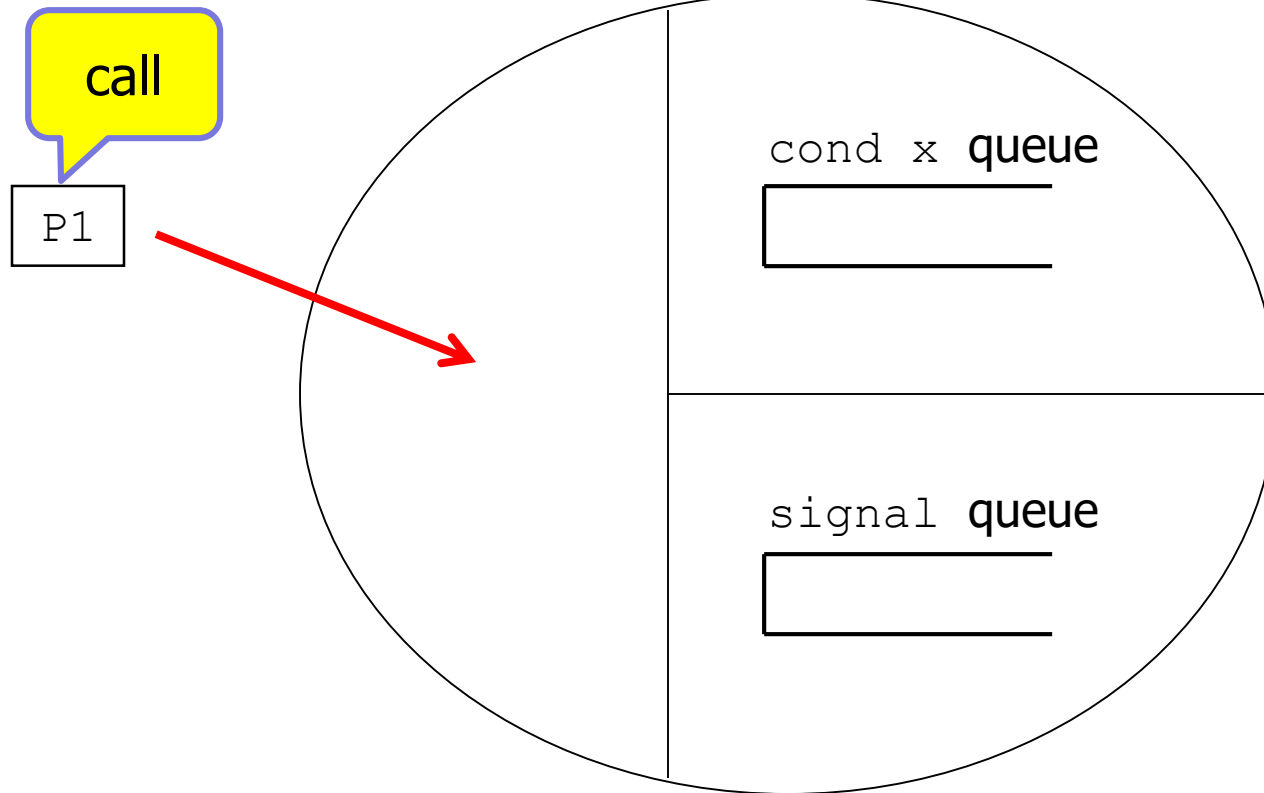


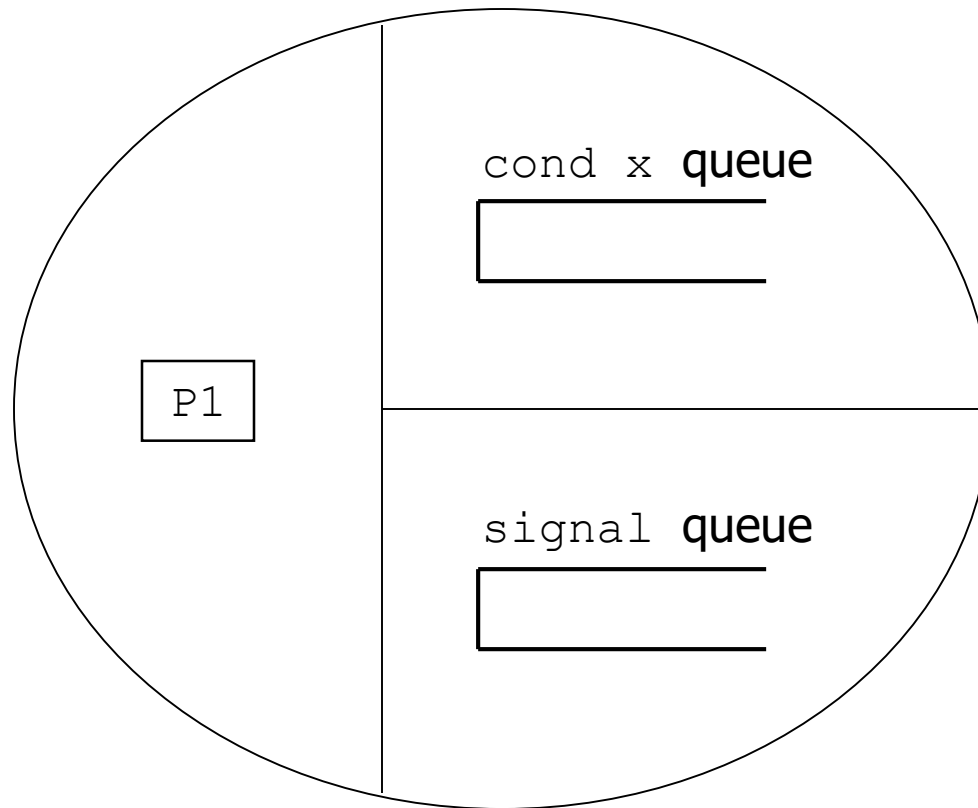
signal queue

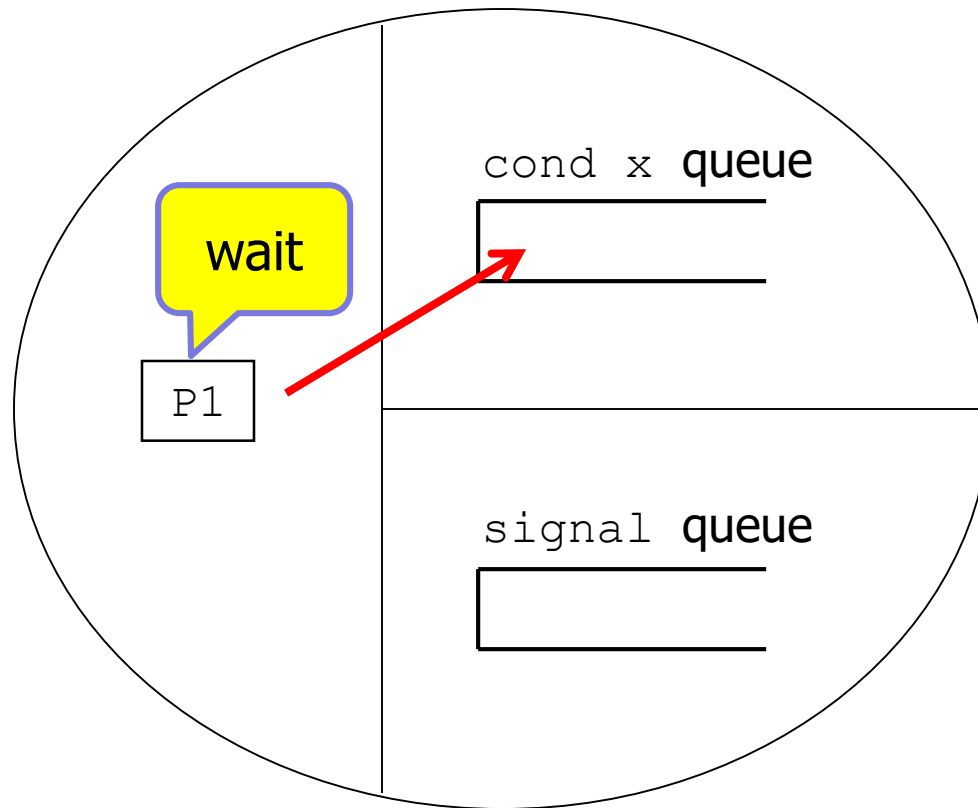


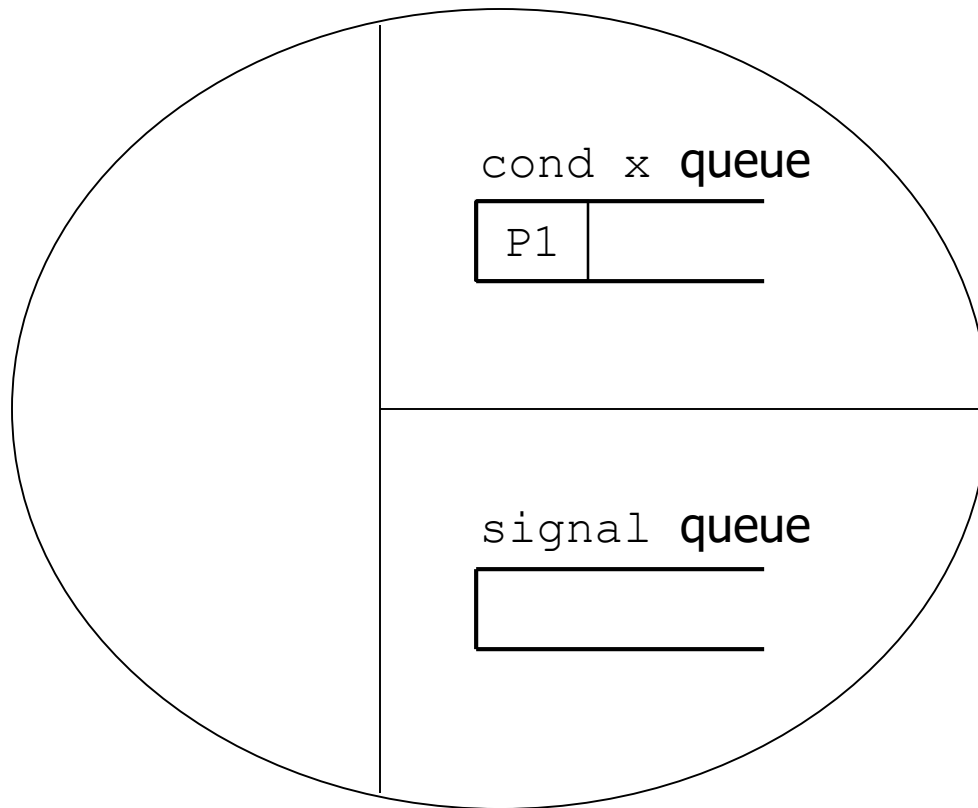
νήμα που εκτελεί
κώδικα του ελεγκτή

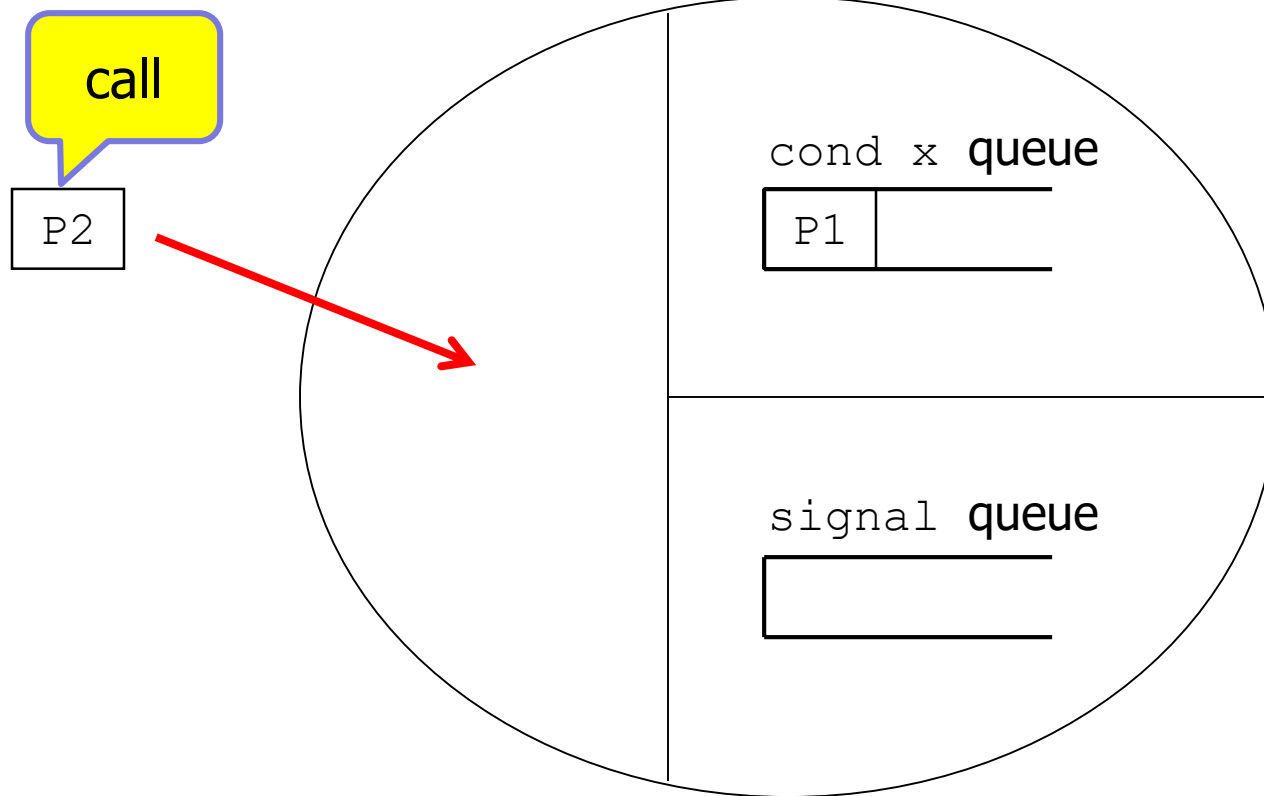
ουρά νημάτων που περιμένουν,
λόγω signal&block / signal&continue,
να ελευθερωθεί ο ελεγκτής

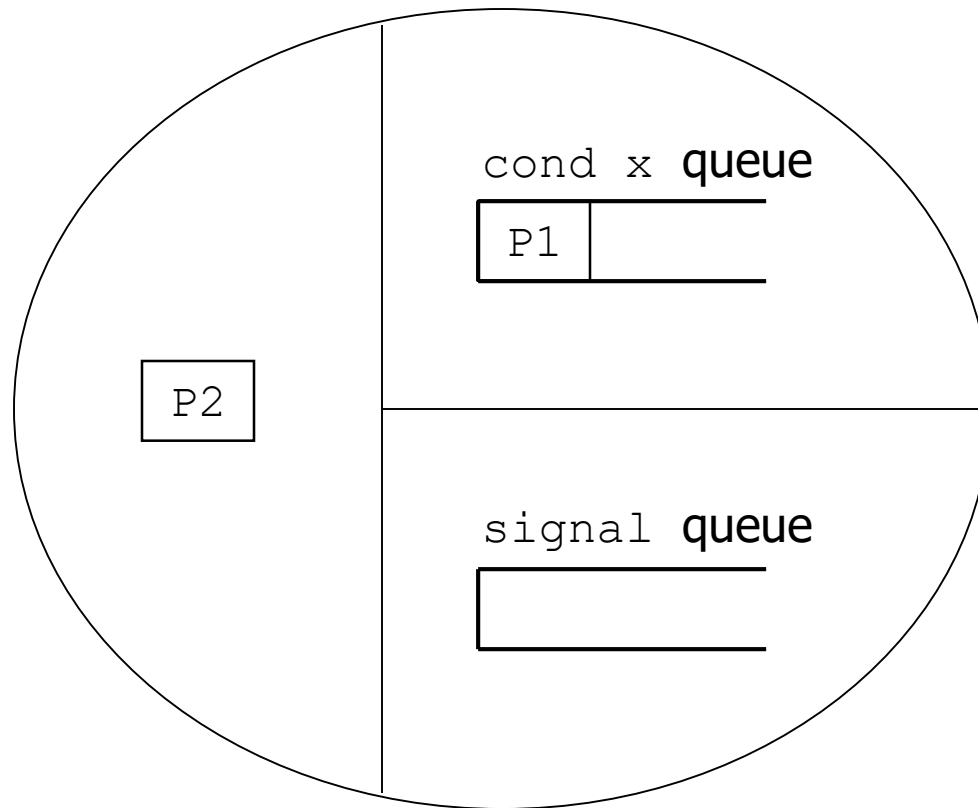




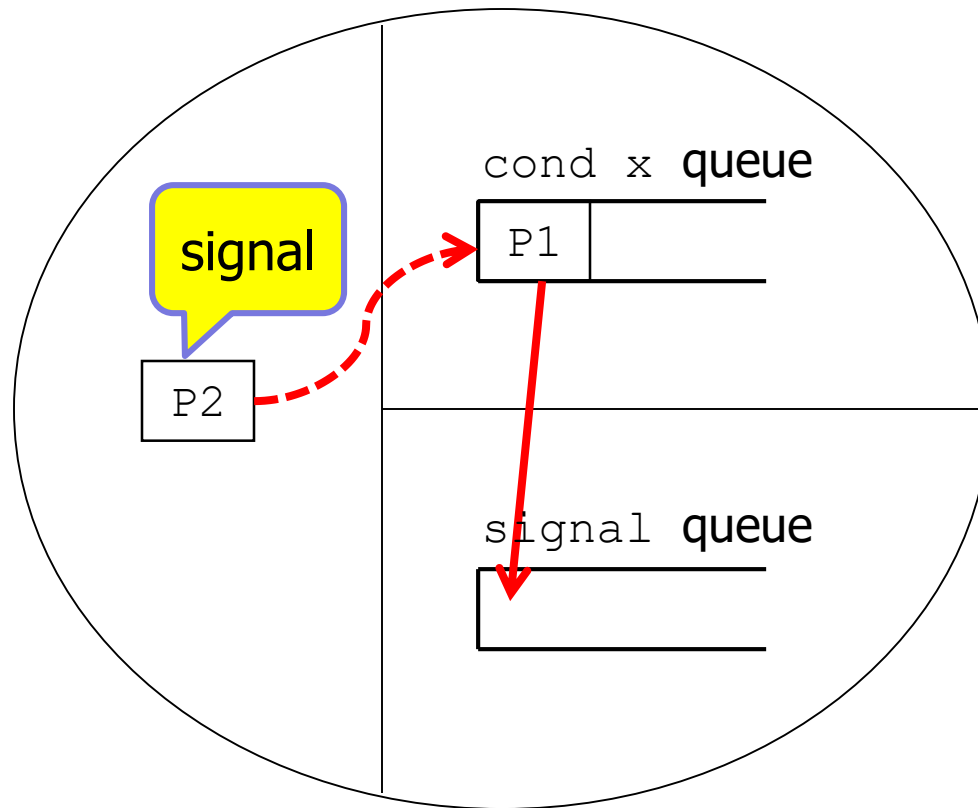




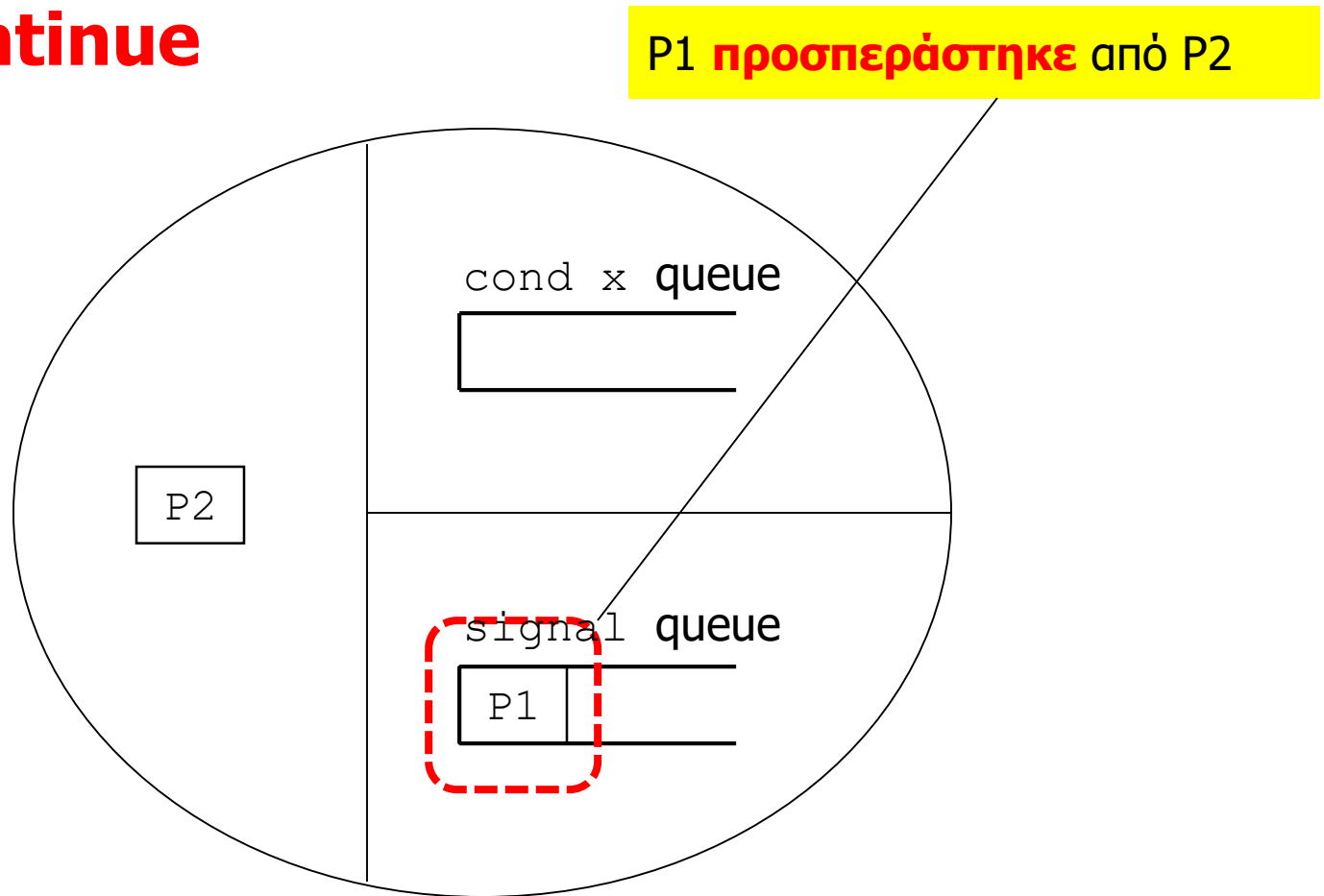




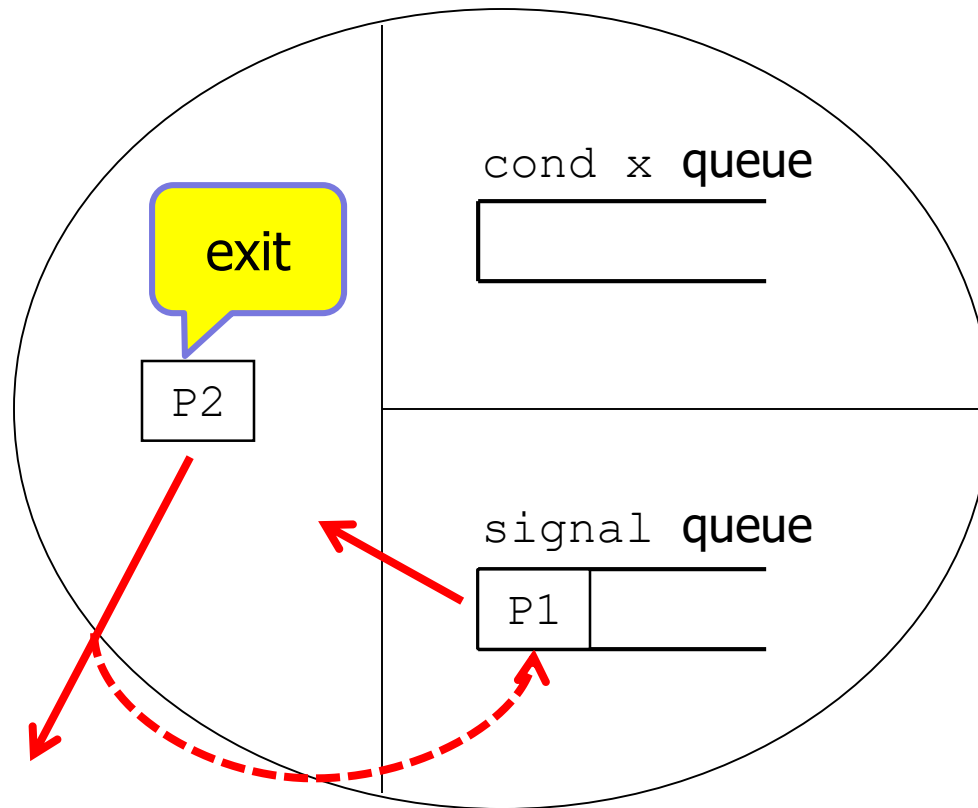
signal-continue



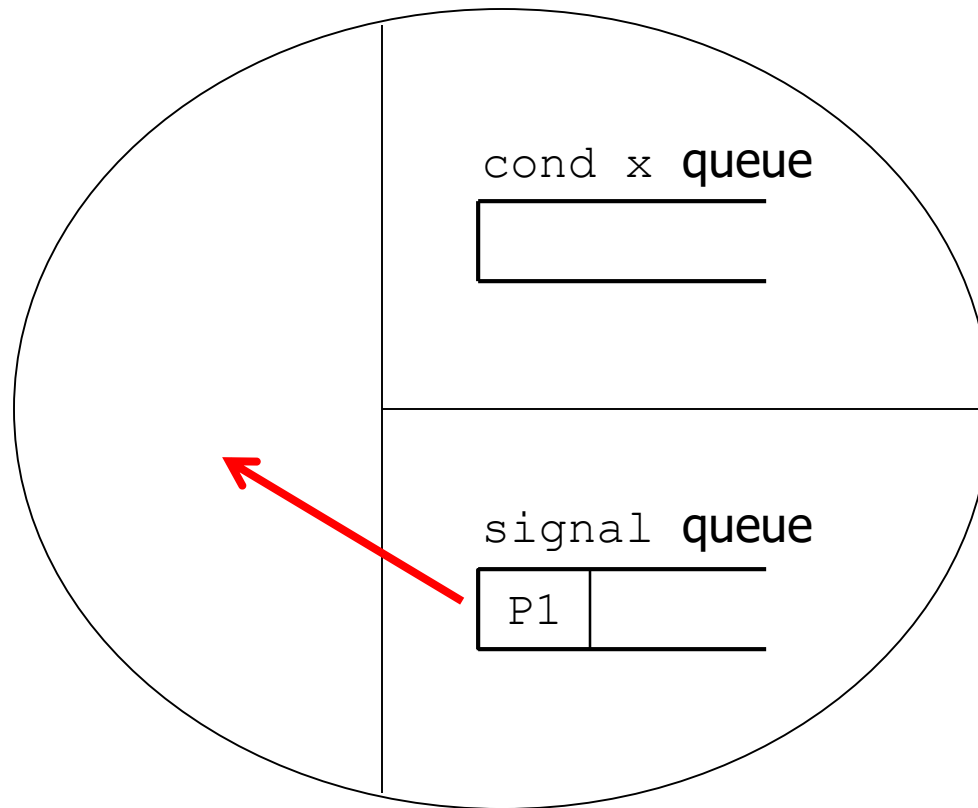
signal-continue



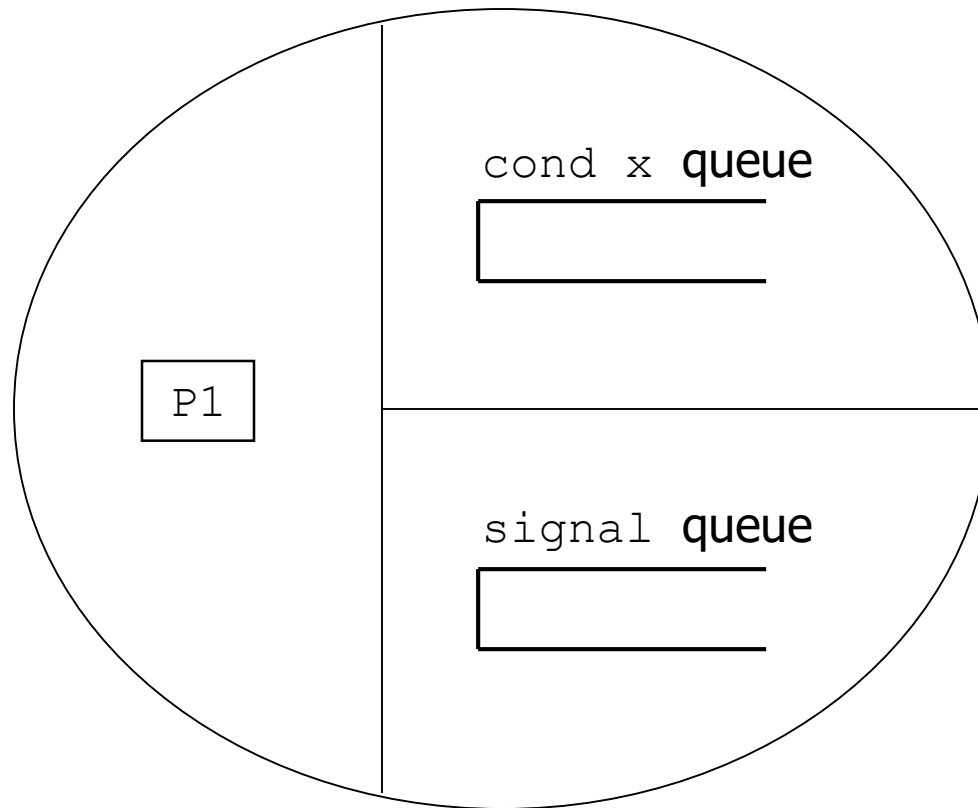
signal-continue



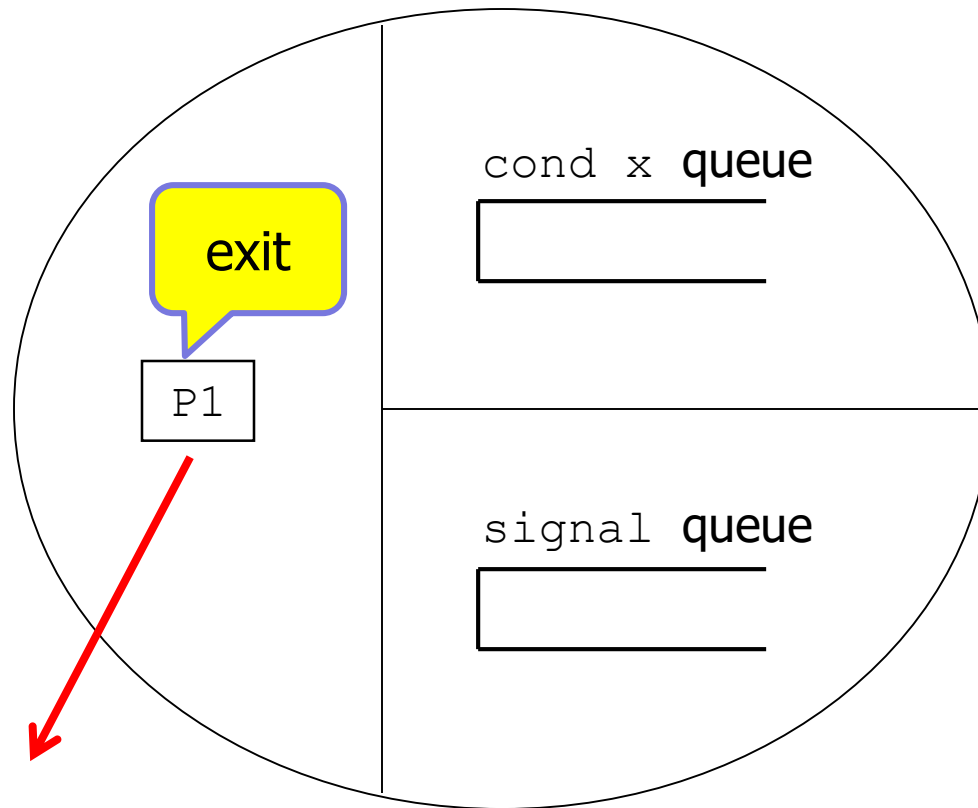
signal-continue



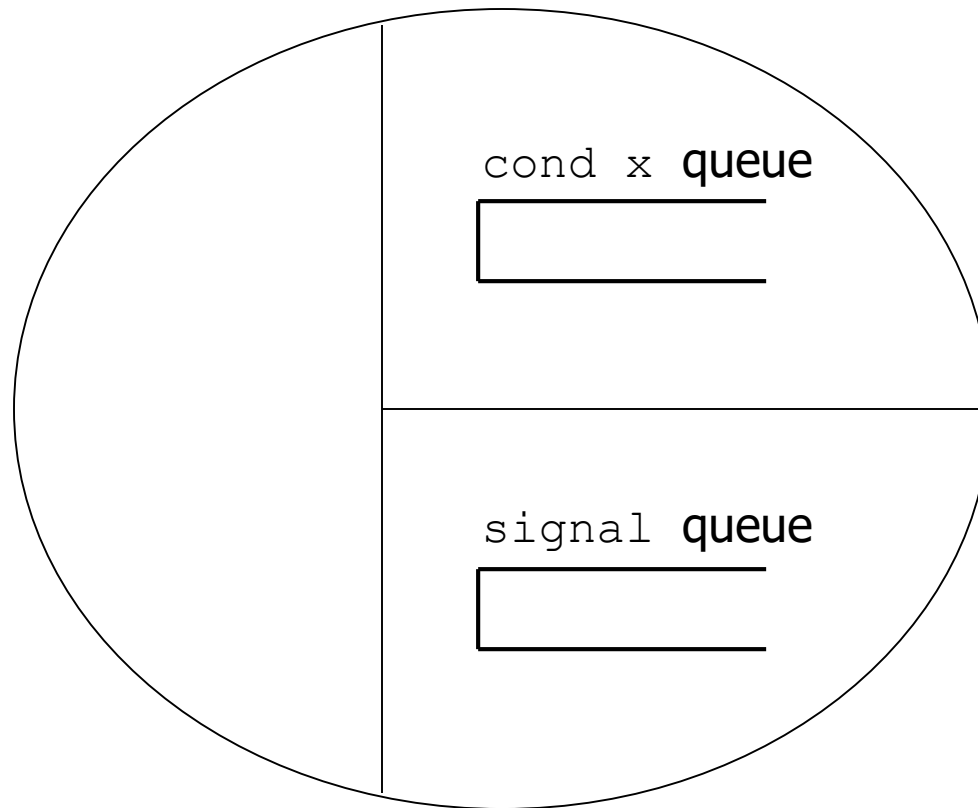
signal-continue

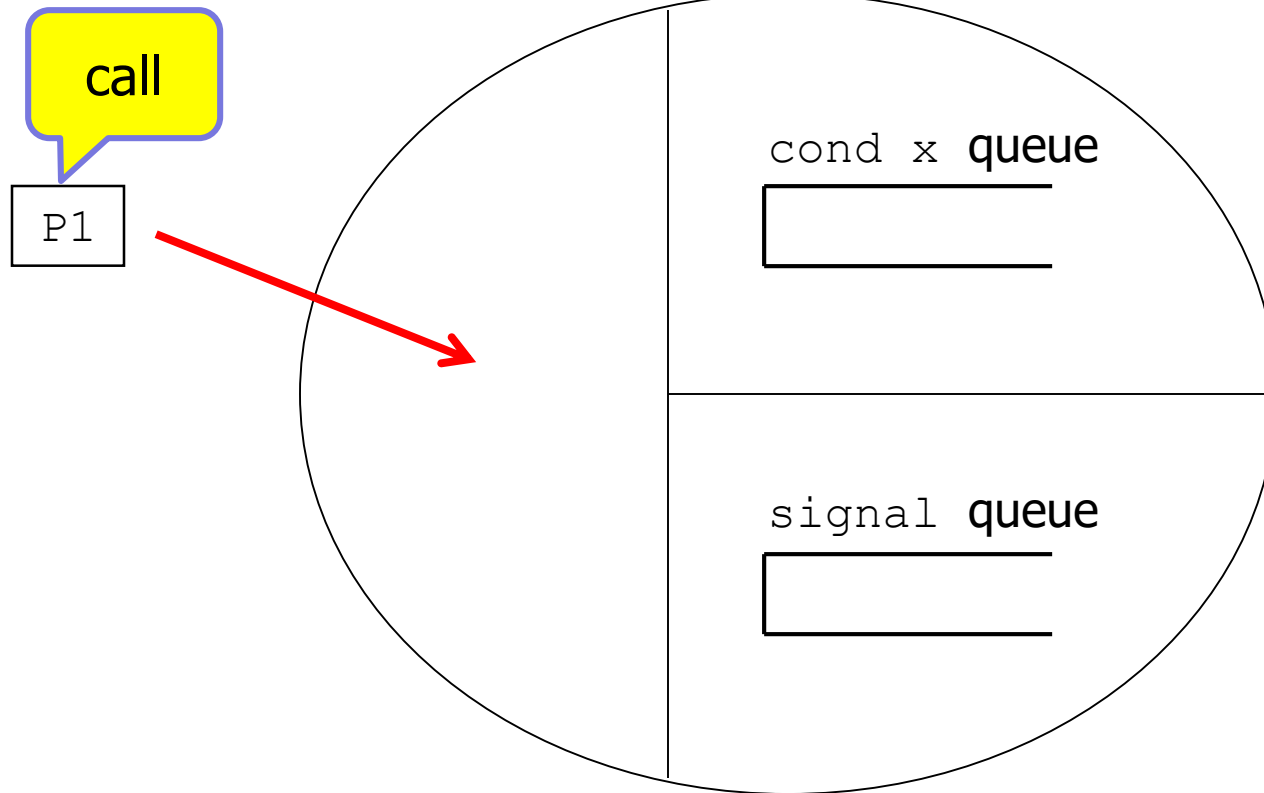


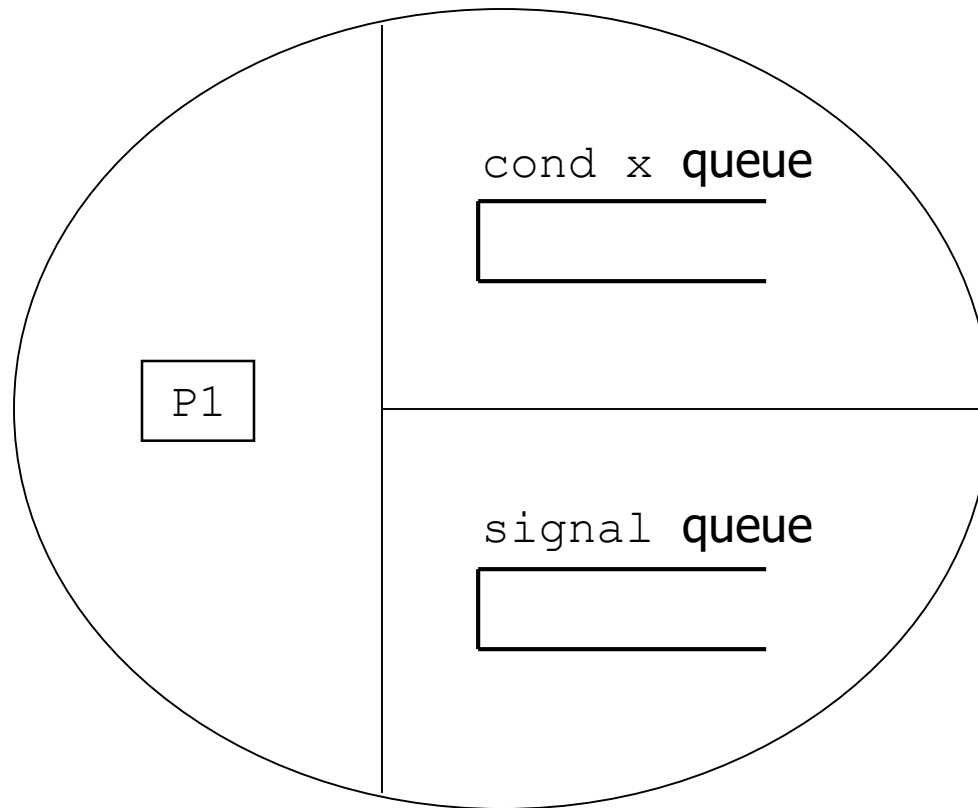
signal-continue

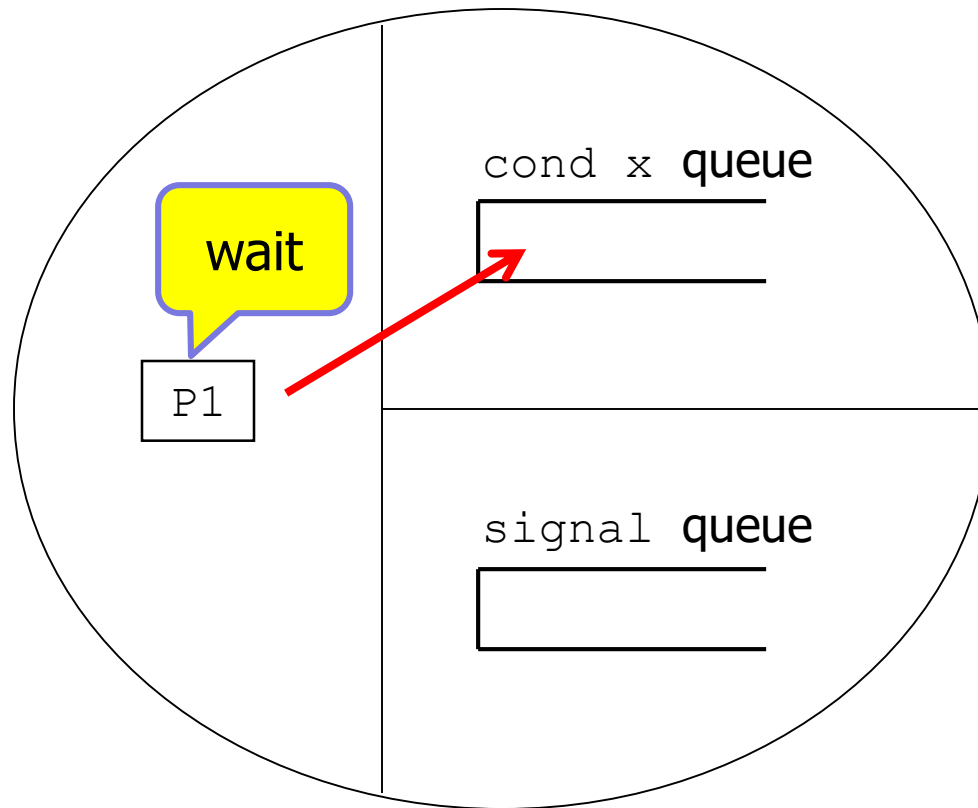


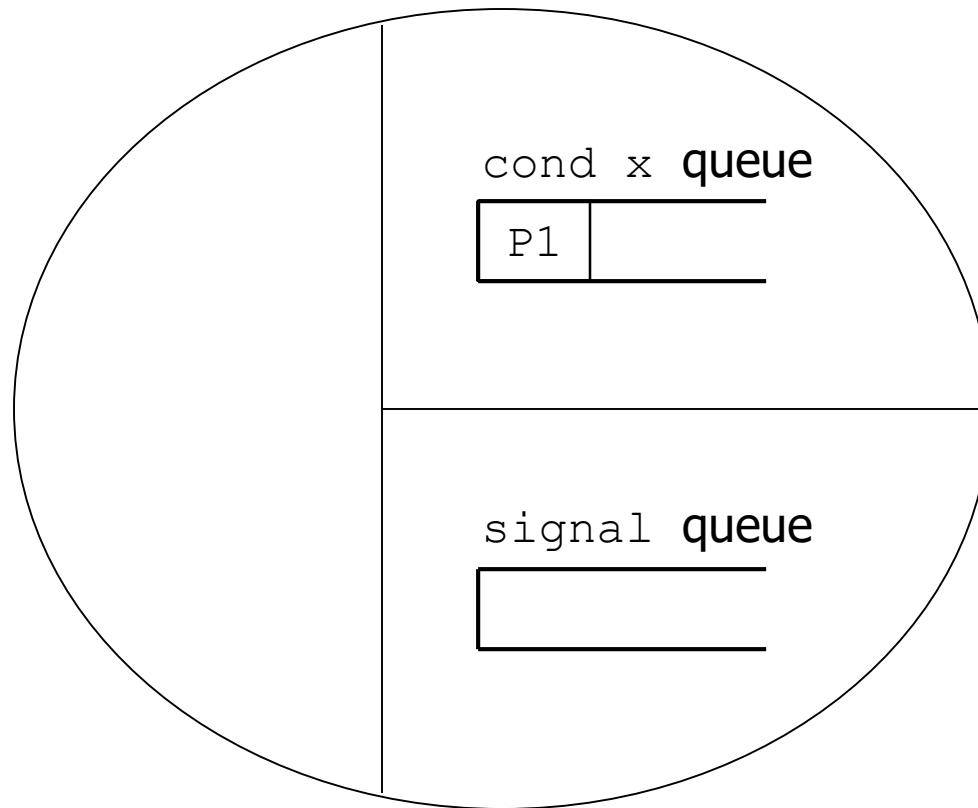
signal-continue

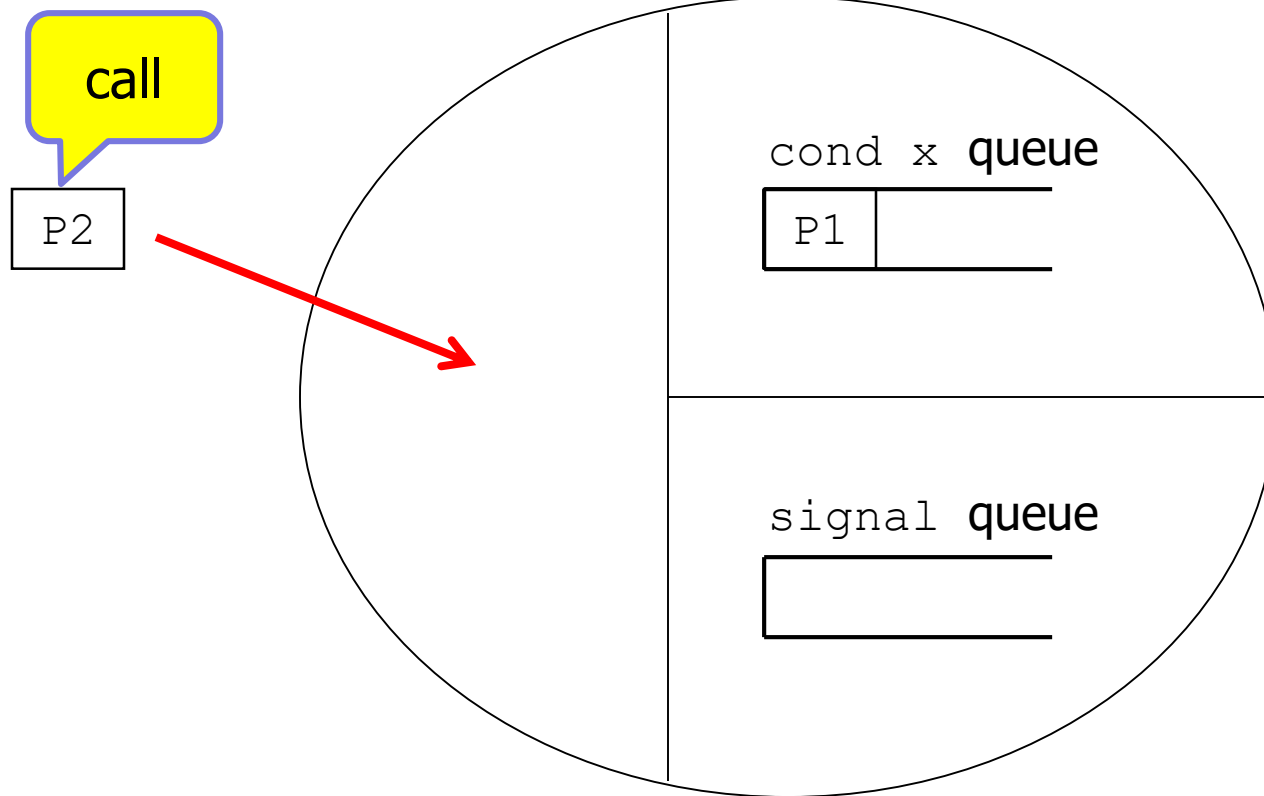


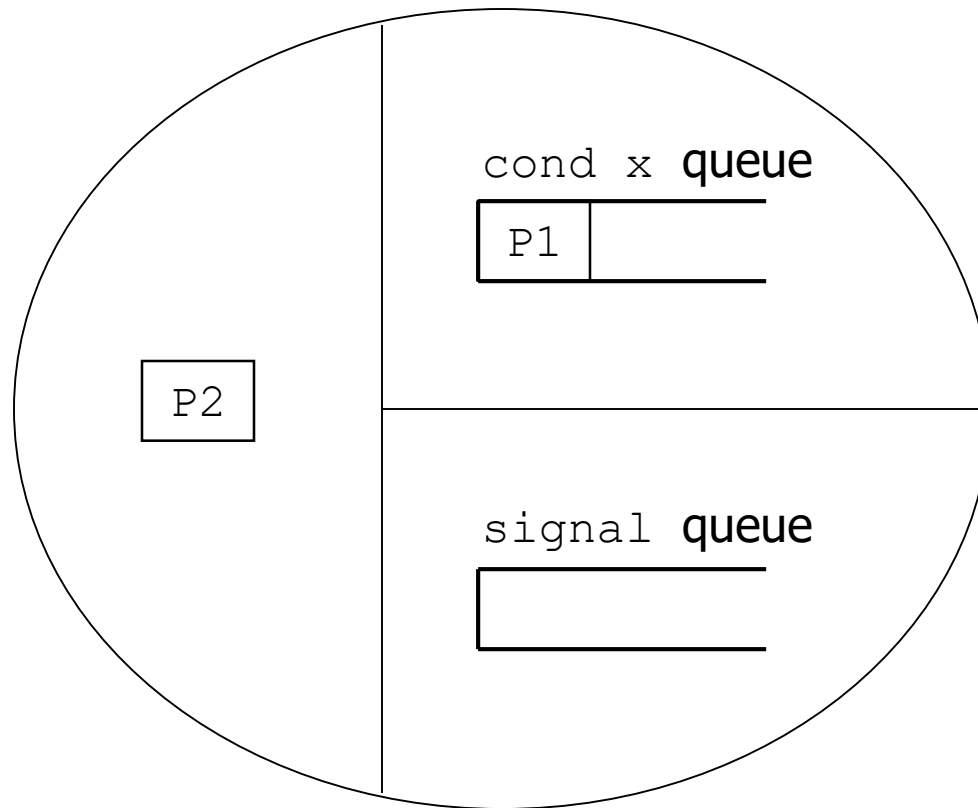




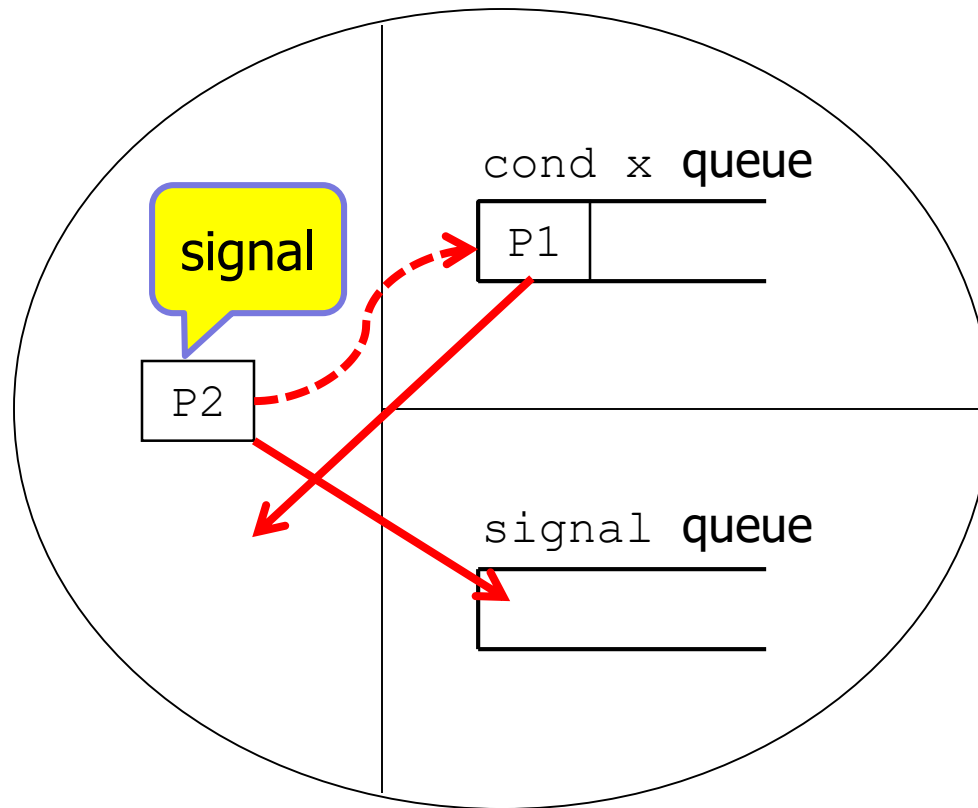




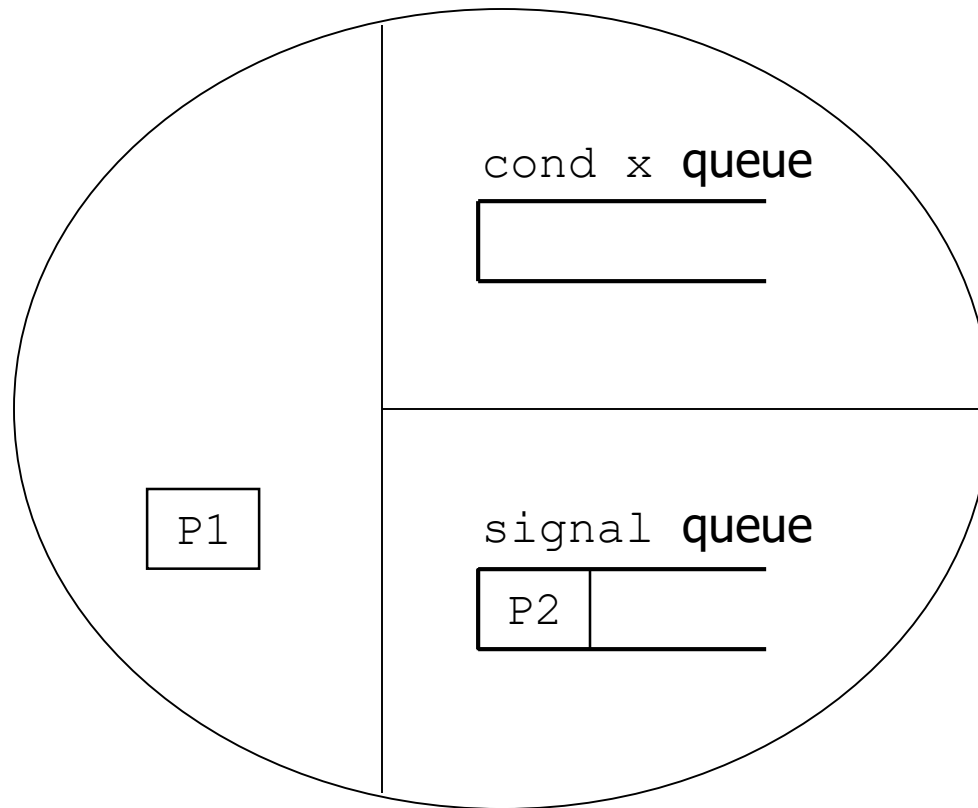




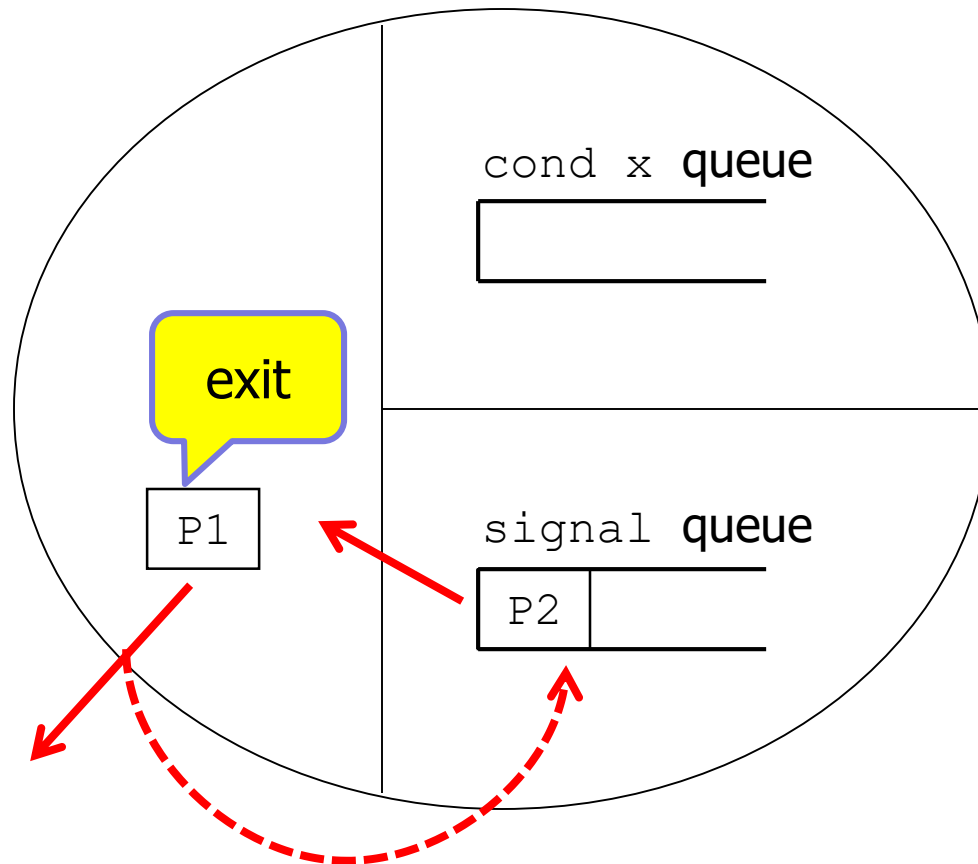
signal-block



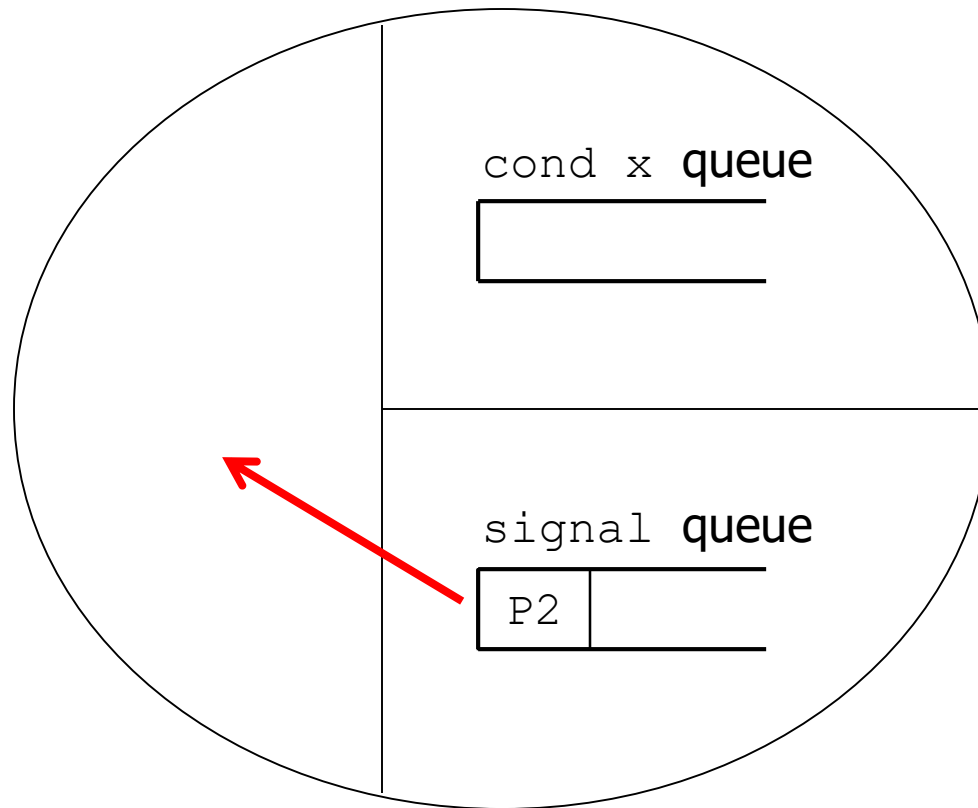
signal-block



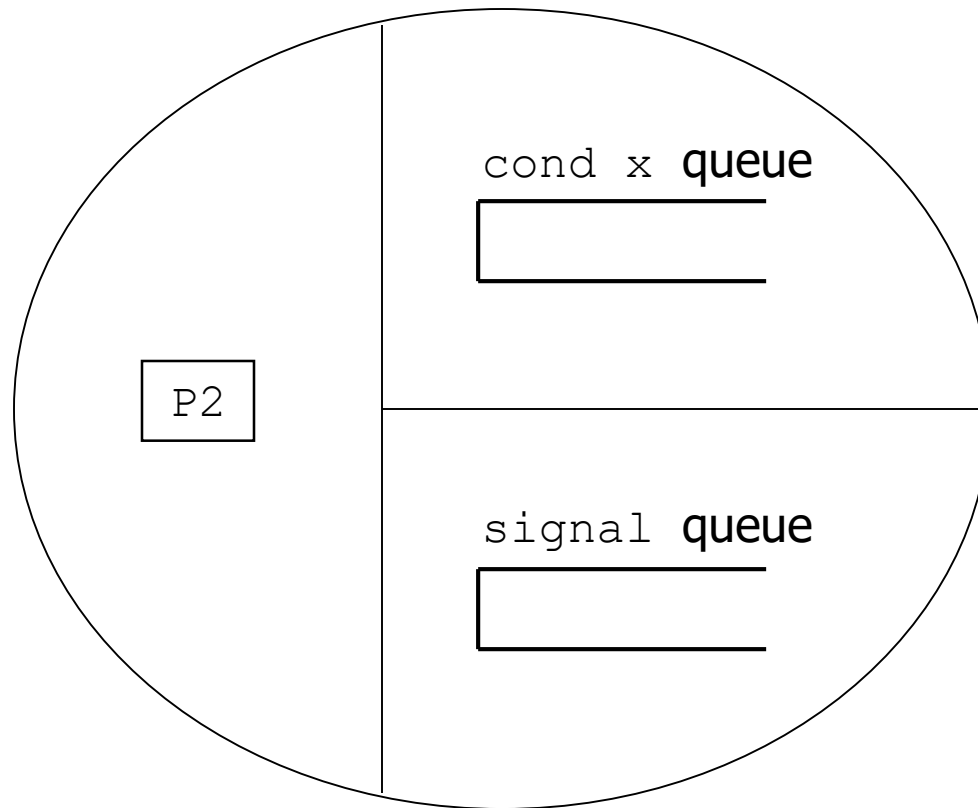
signal-block



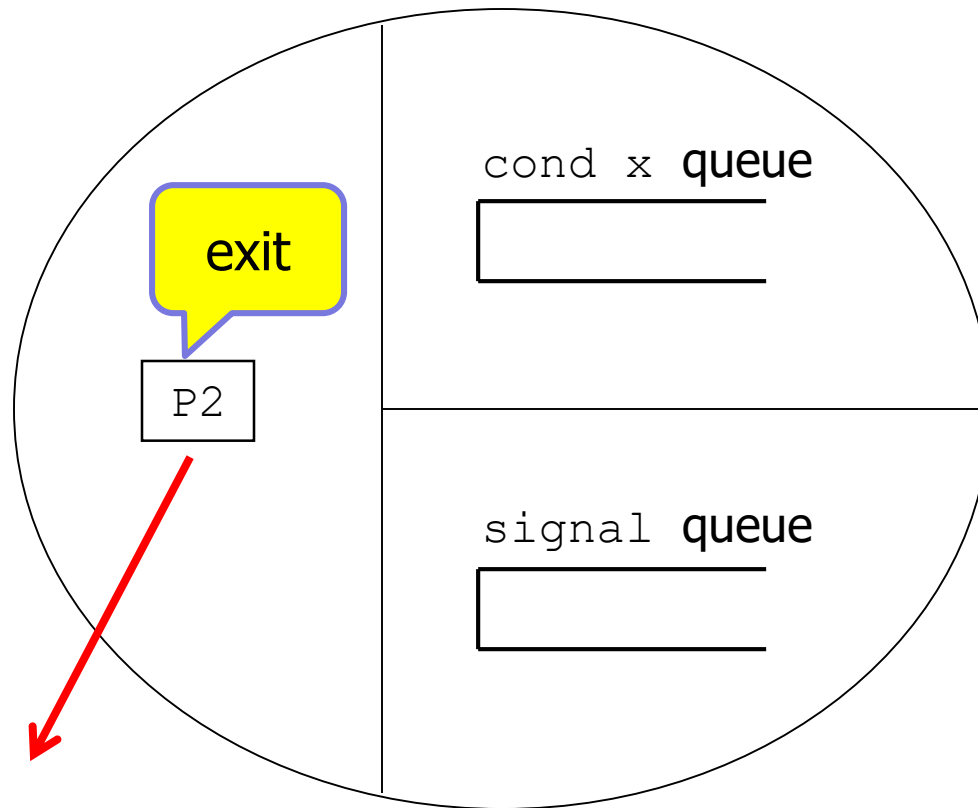
signal-block



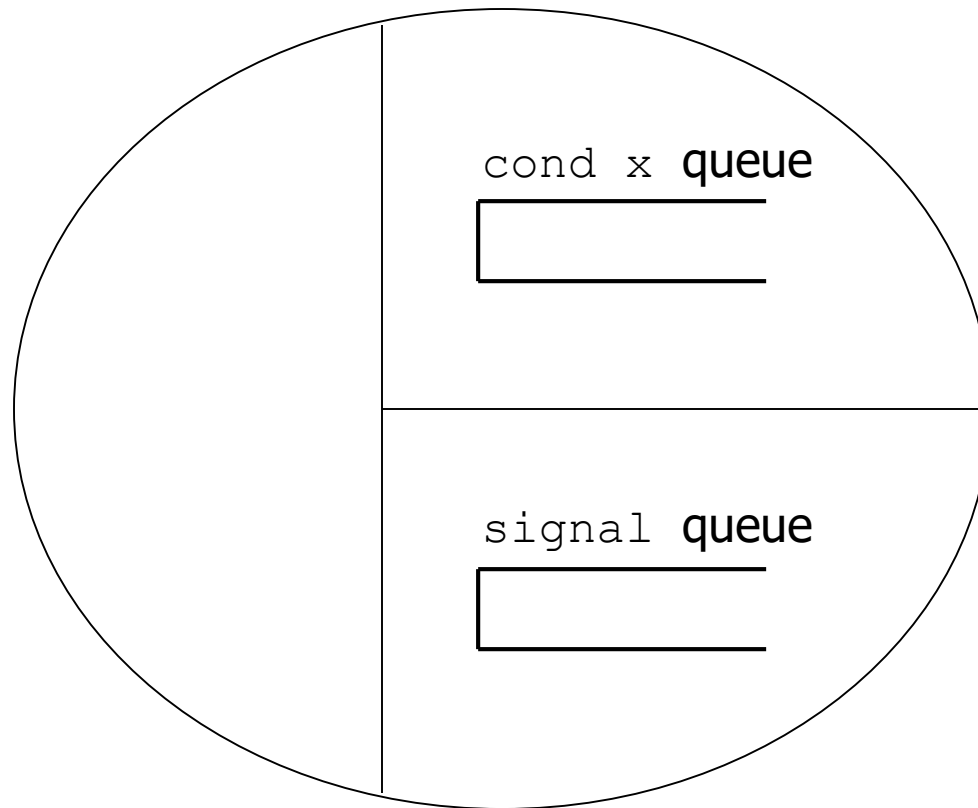
signal-block



signal-block



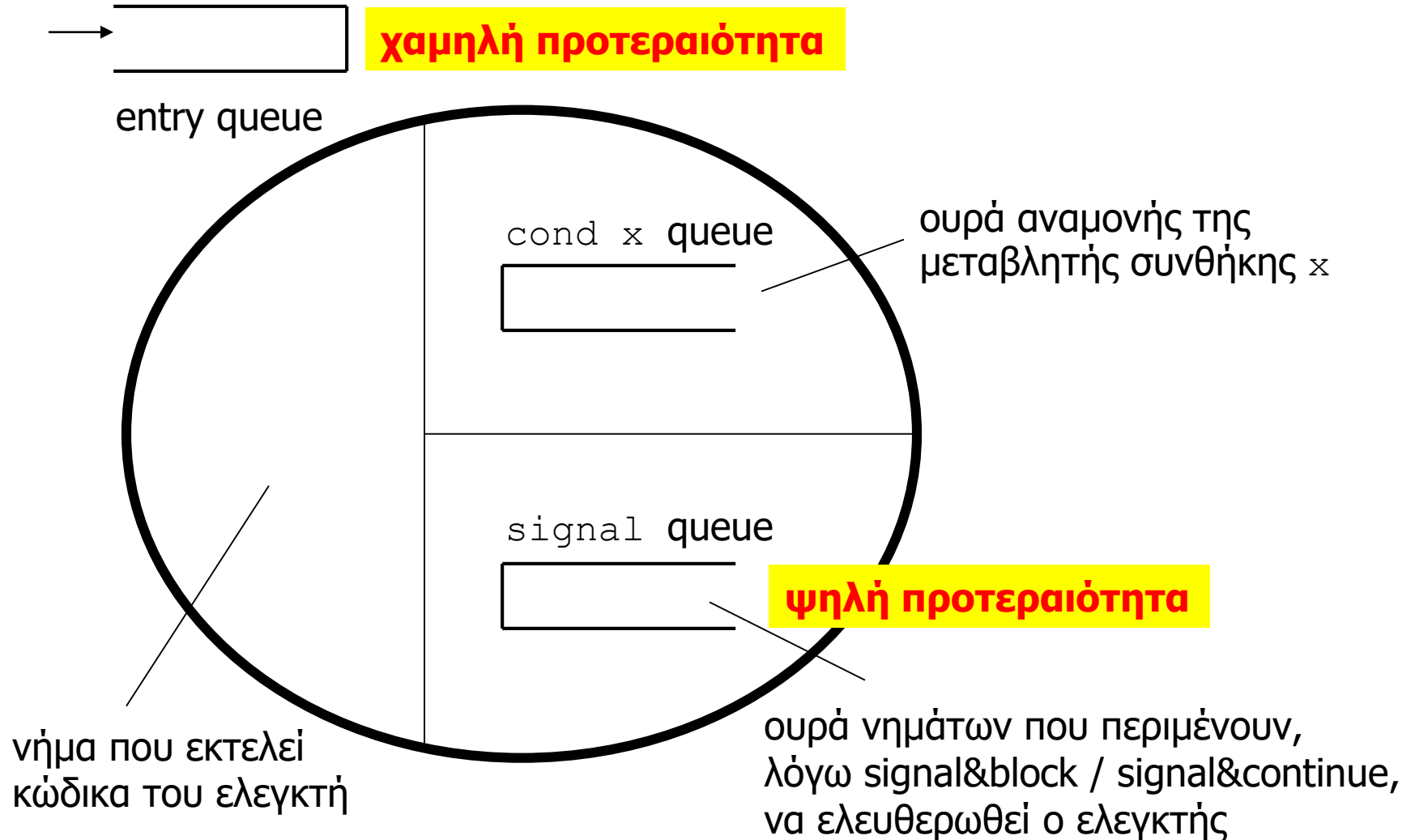
signal-block



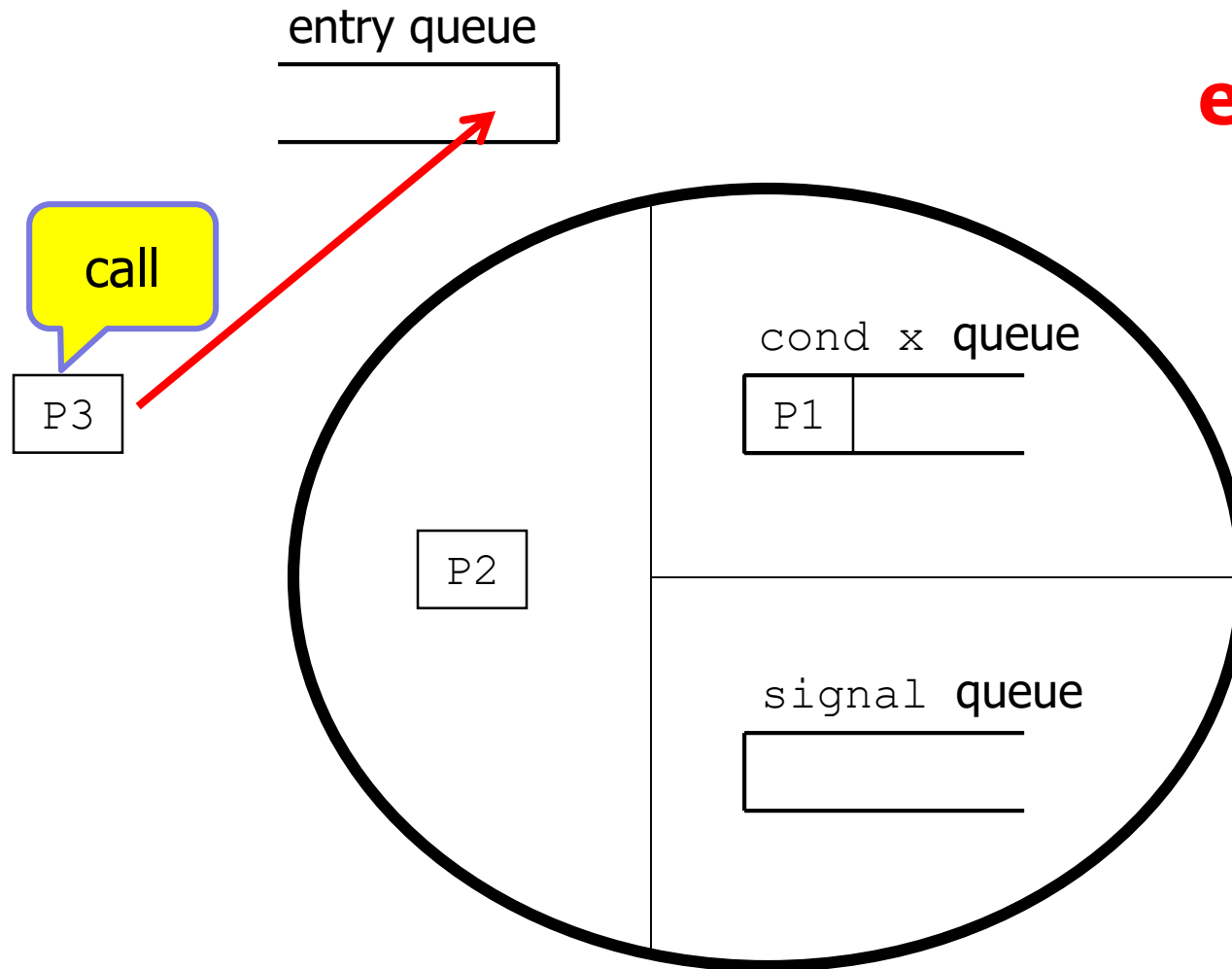
Προτεραιότητες (2)

- Τι γίνεται με τα νήματα που έχουν μπλοκάρει **έξω** από τον ελεγκτή, περιμένοντας να τους δοθεί άδεια να προχωρήσουν με την εκτέλεση μιας λειτουργίας;
- **Eggshell:** προτεραιότητα έχουν τα νήματα που ήδη έχουν αρχίσει την εκτέλεση τους μέσα στον ελεγκτή
 - λογική FIFO, δεν υπάρχουν «περίεργα» προσπεράσματα
- **Open:** τα νήματα έχουν την ίδια προτεραιότητα
 - ένα νήμα που αφυπνίζει/αφυπνίζεται μέσω `signal`, μπορεί **προσπεραστεί** από ένα άλλο νήμα που περιμένει να μπει στον ελεγκτή (δεν έχει αρχίσει ακόμα να εκτελεί κώδικα μέσα στο πλαίσιο του ελεγκτή)

Eggshell



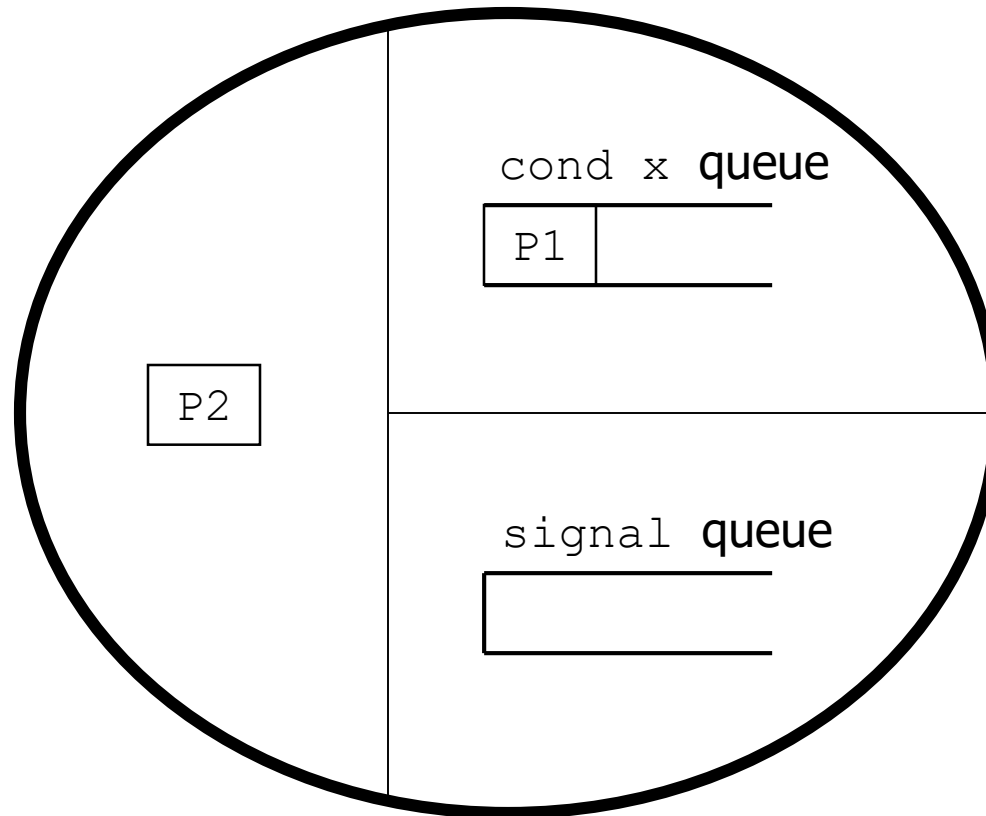
eggshell



entry queue



eggshell

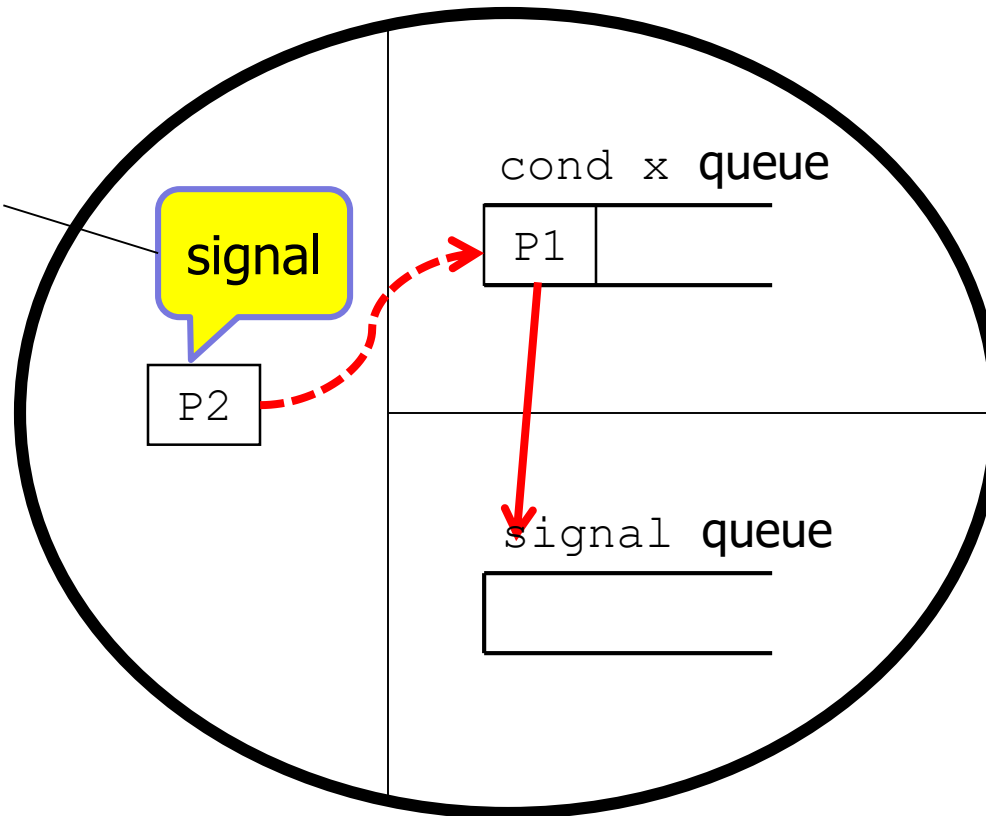


entry queue



eggshell

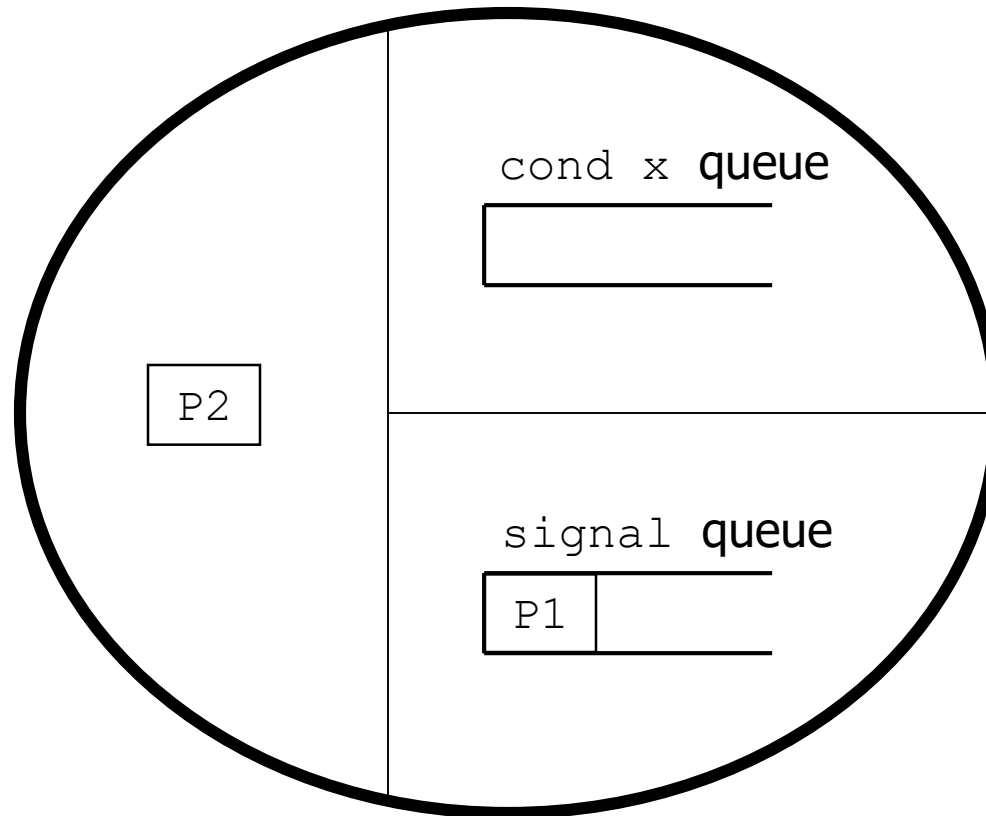
signal-continue



entry queue



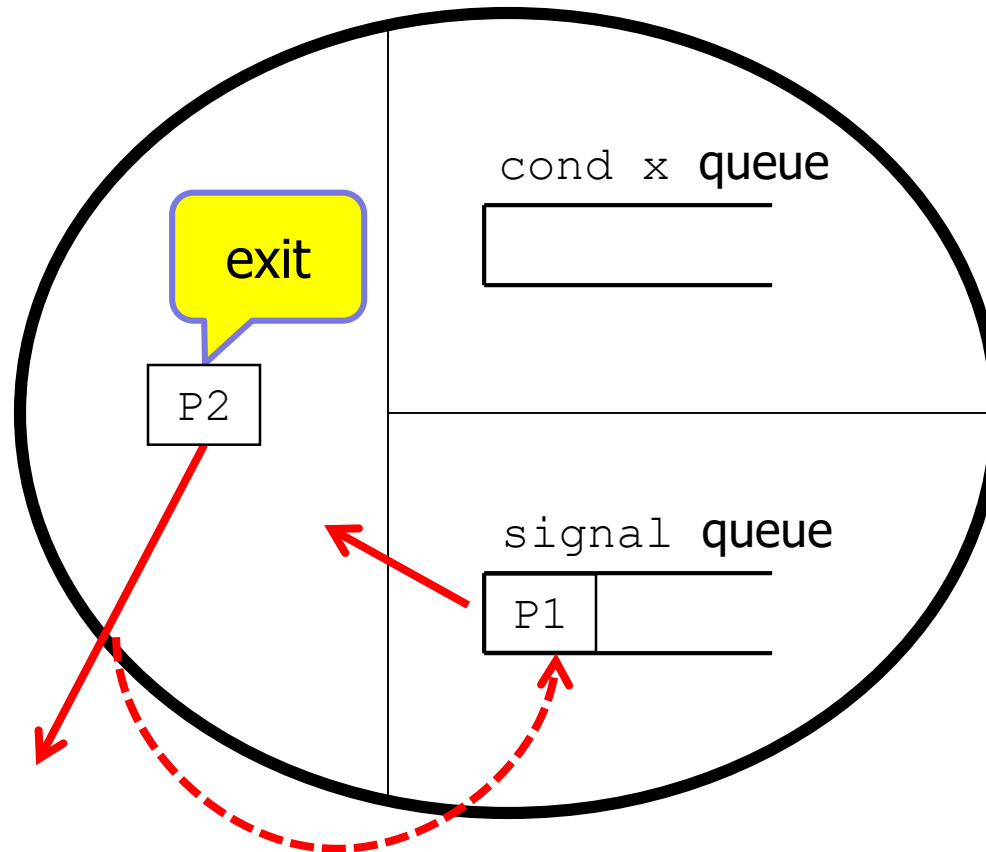
eggshell



entry queue



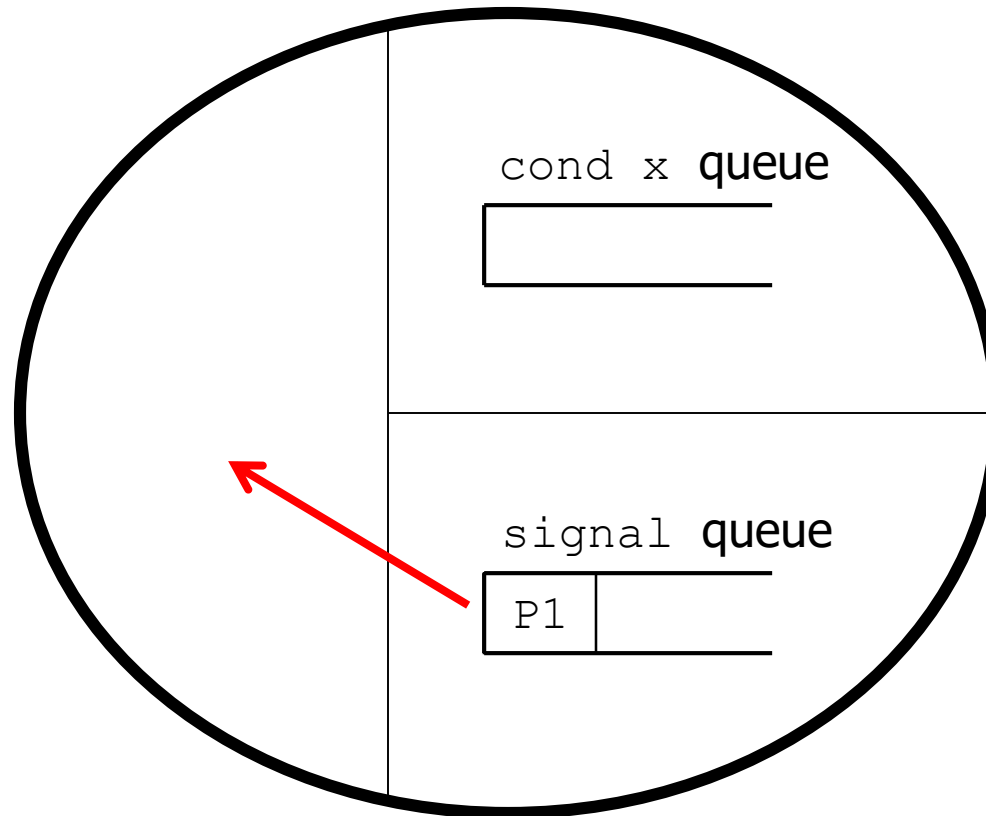
eggshell



entry queue



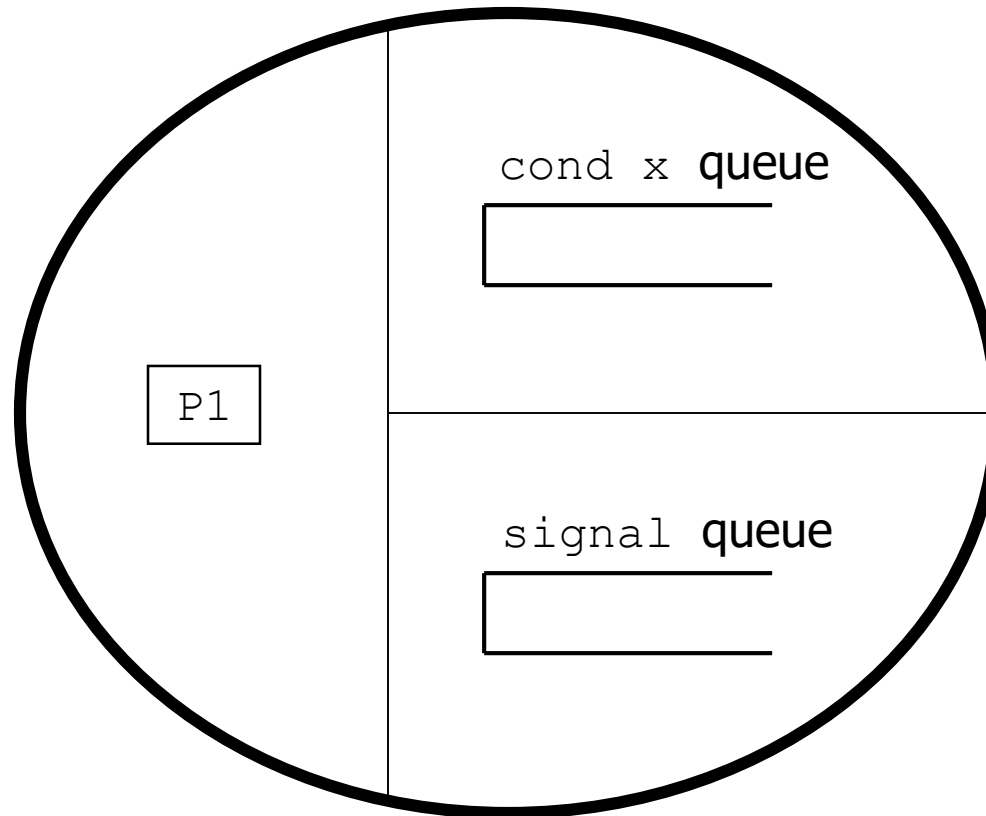
eggshell



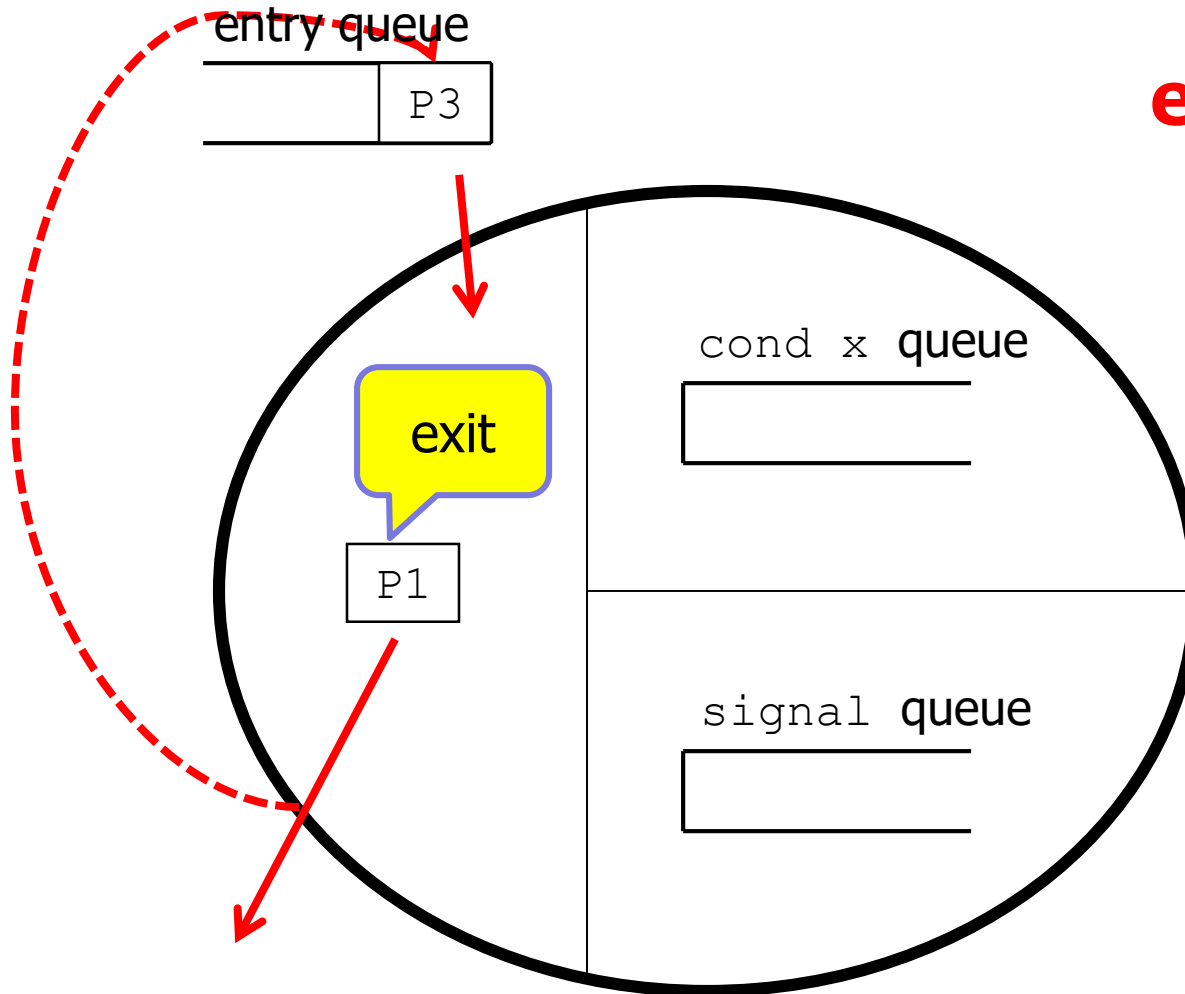
entry queue



eggshell



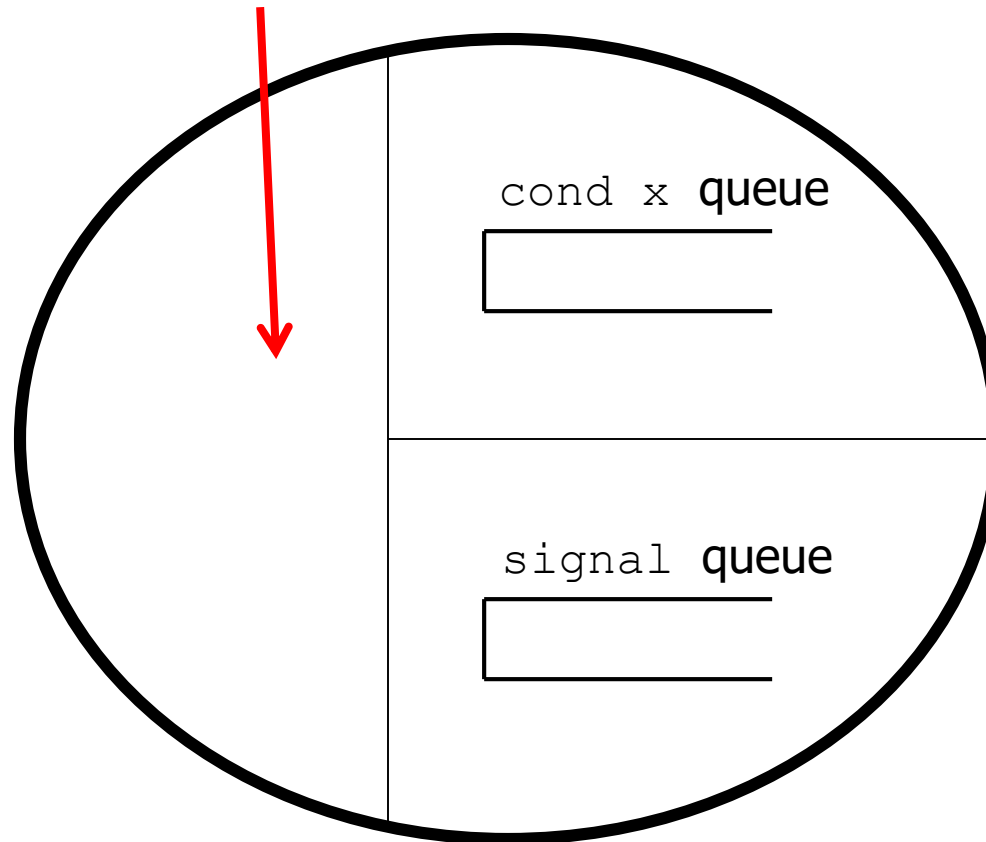
eggshell



entry queue



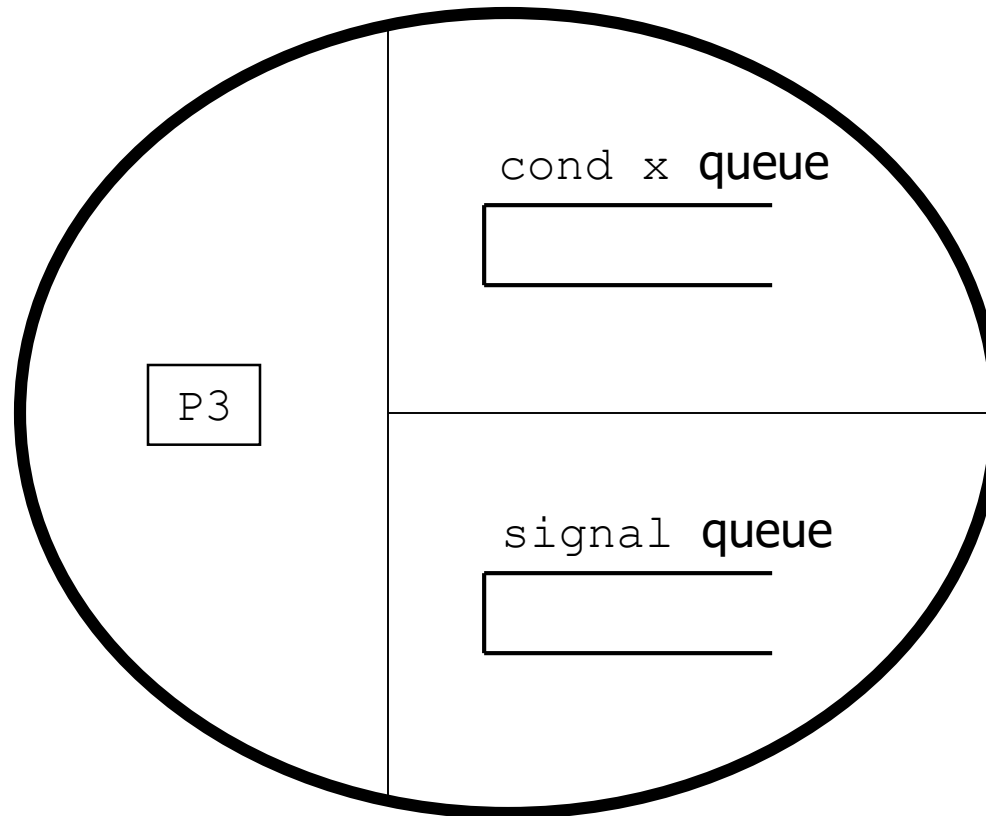
eggshell



entry queue



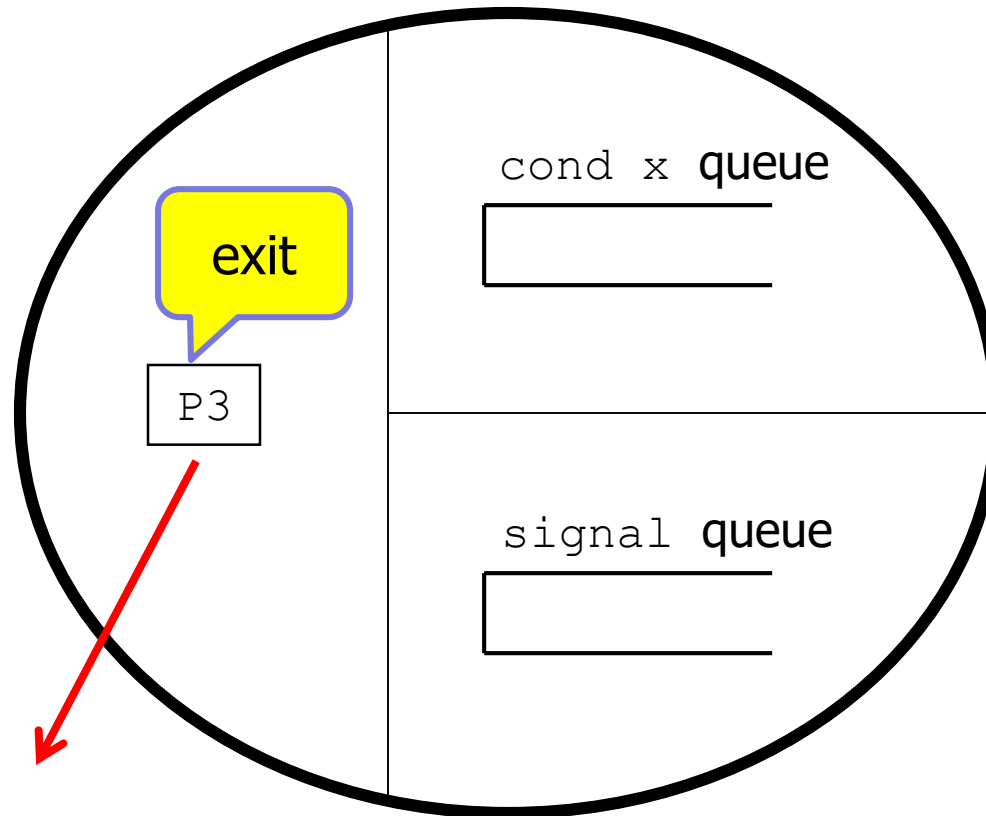
eggshell



entry queue



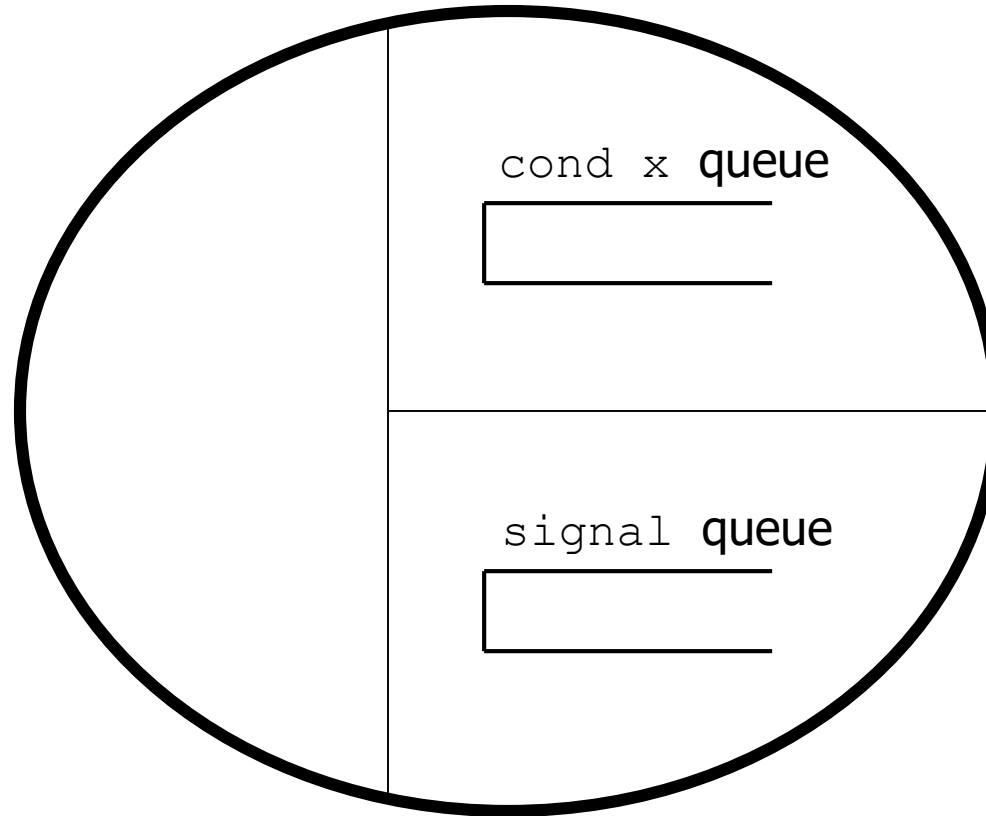
eggshell



entry queue



eggshell

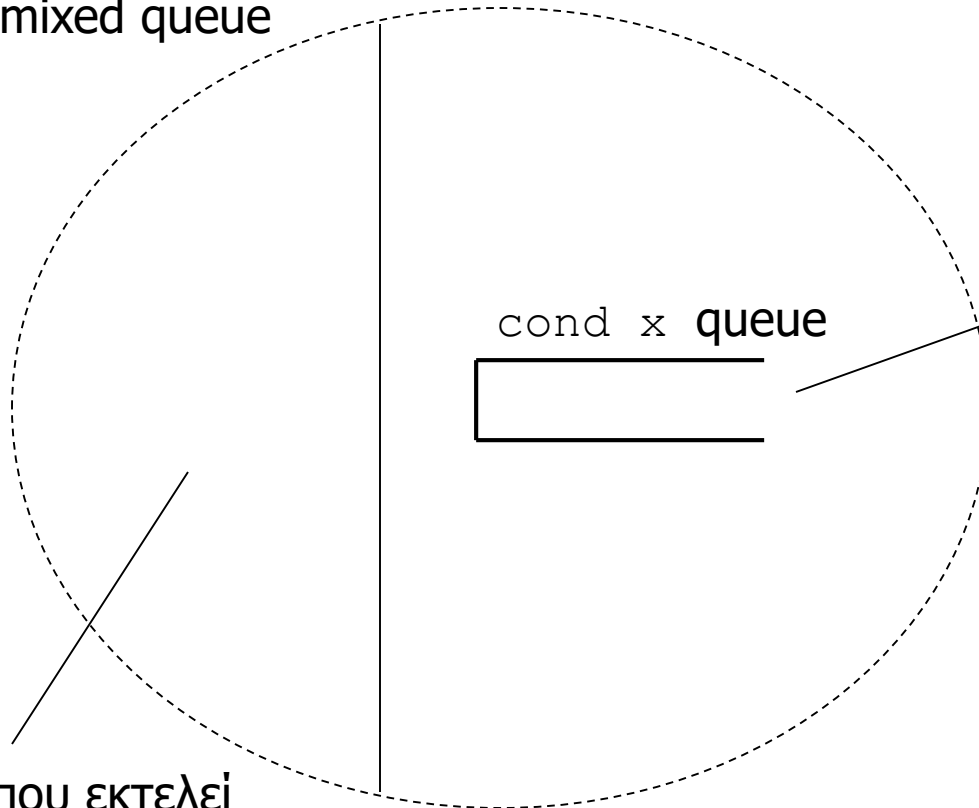


Open

κοινή ουρά (α) για τα νήματα που θέλουν να μπουν στον ελεγκτή **και** (β) για τα νήματα που είχαν μπλοκάρει λόγω signal μέσα στον ελεγκτή

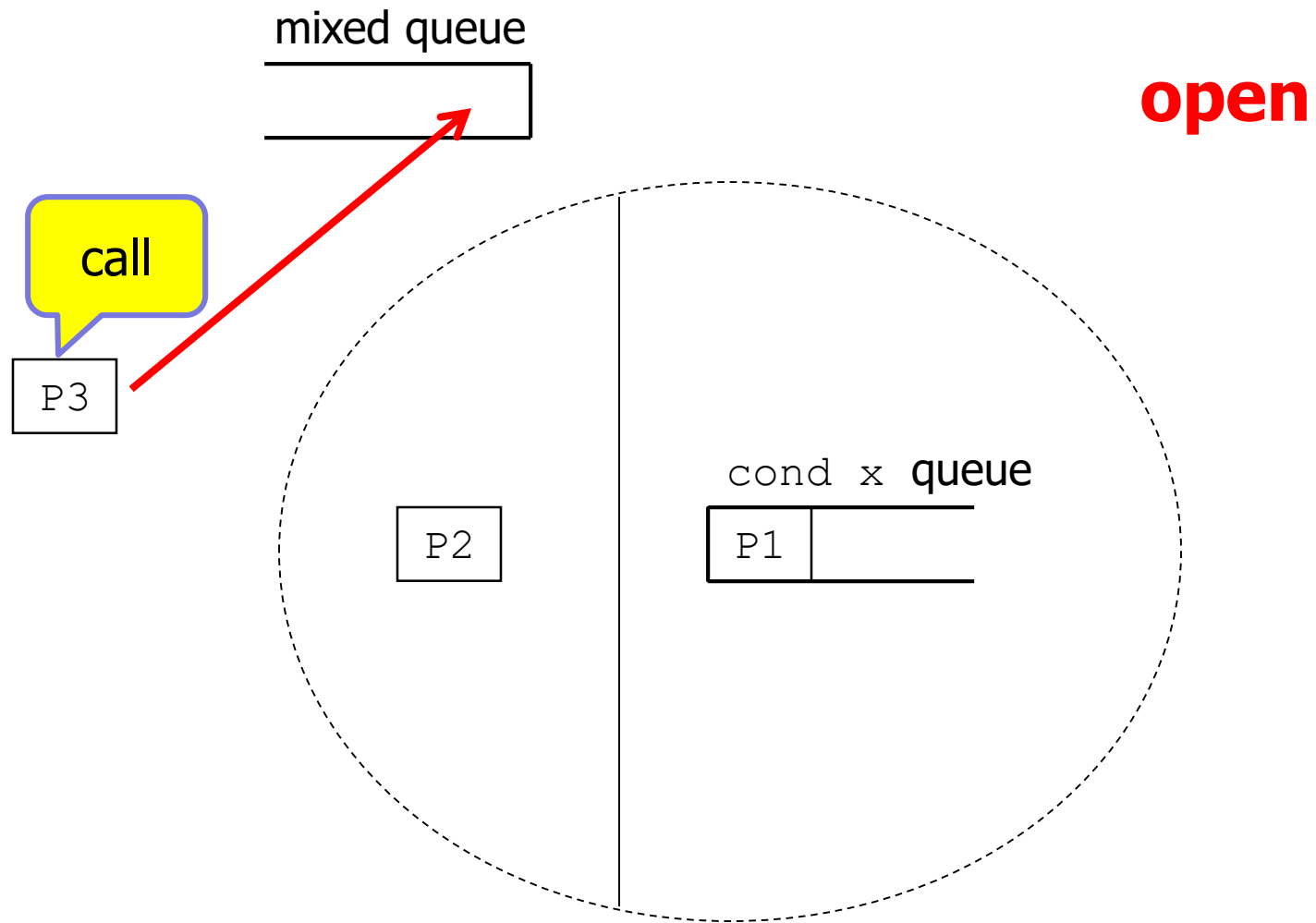


mixed queue



ουρά αναμονής της μεταβλητής συνθήκης x

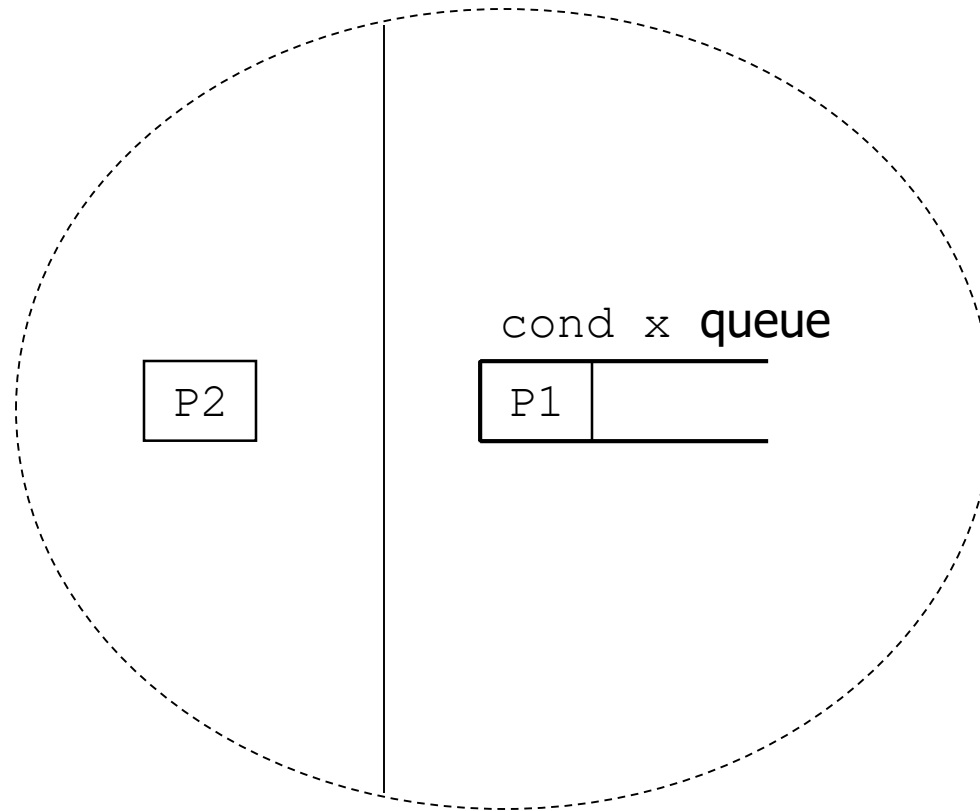
νήμα που εκτελεί
κώδικα του ελεγκτή

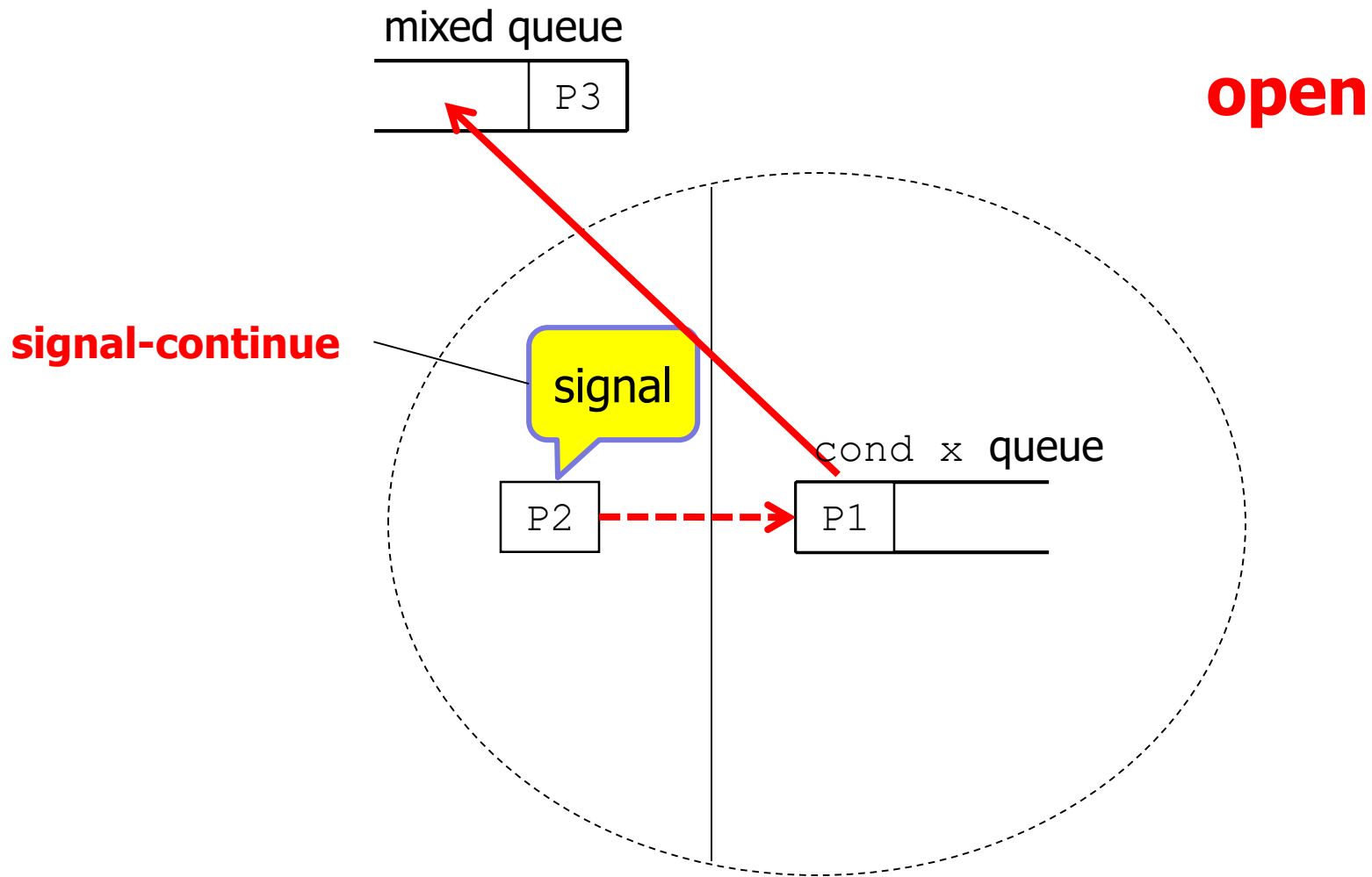


mixed queue

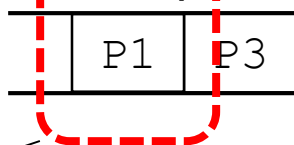


open



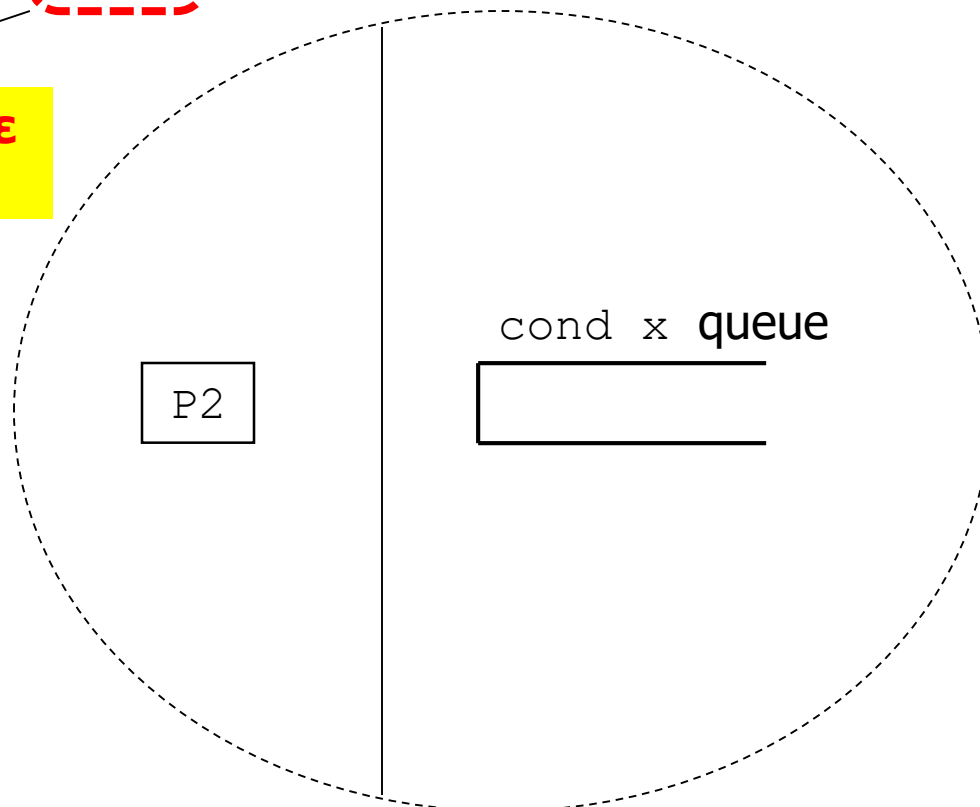


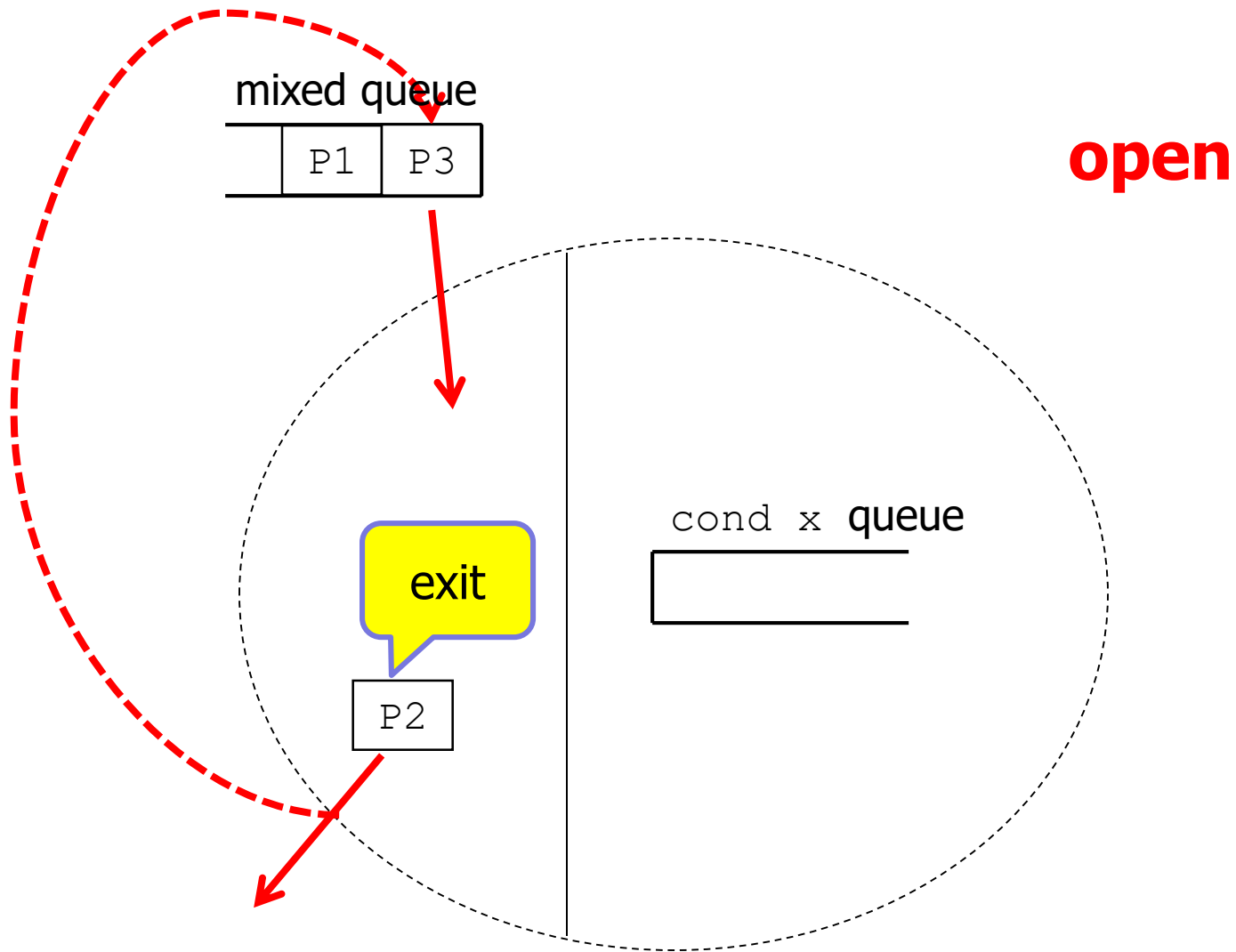
mixed queue



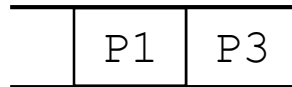
open

**P1 προπεράστηκε
από το P3**

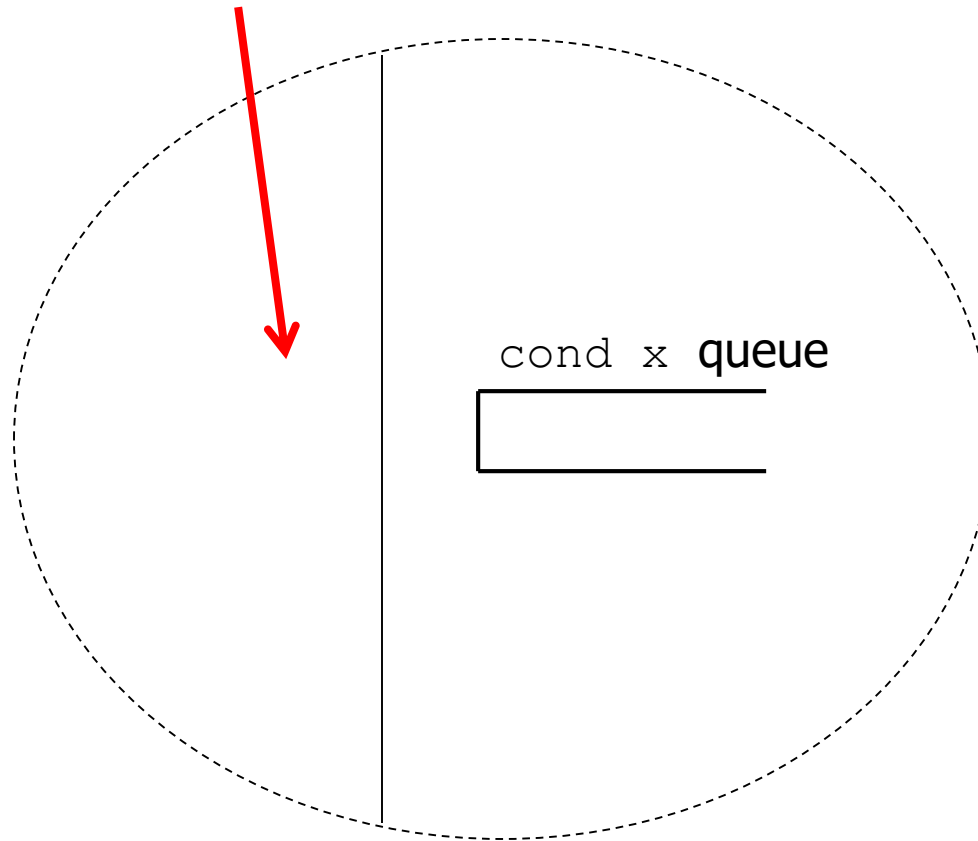




mixed queue



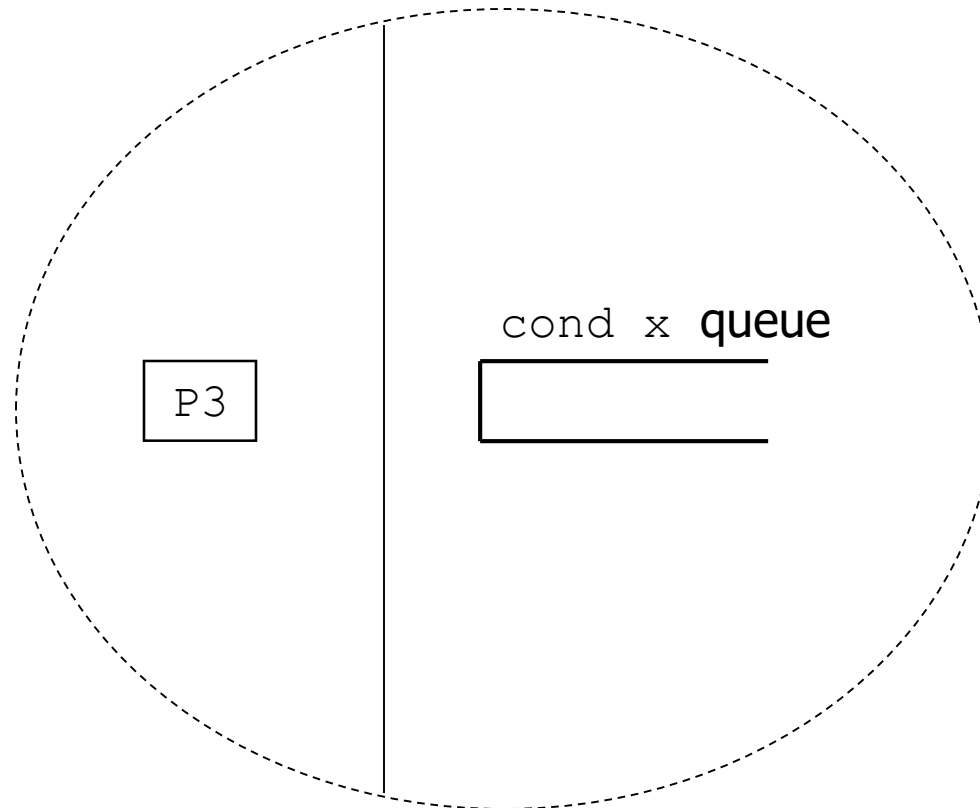
open

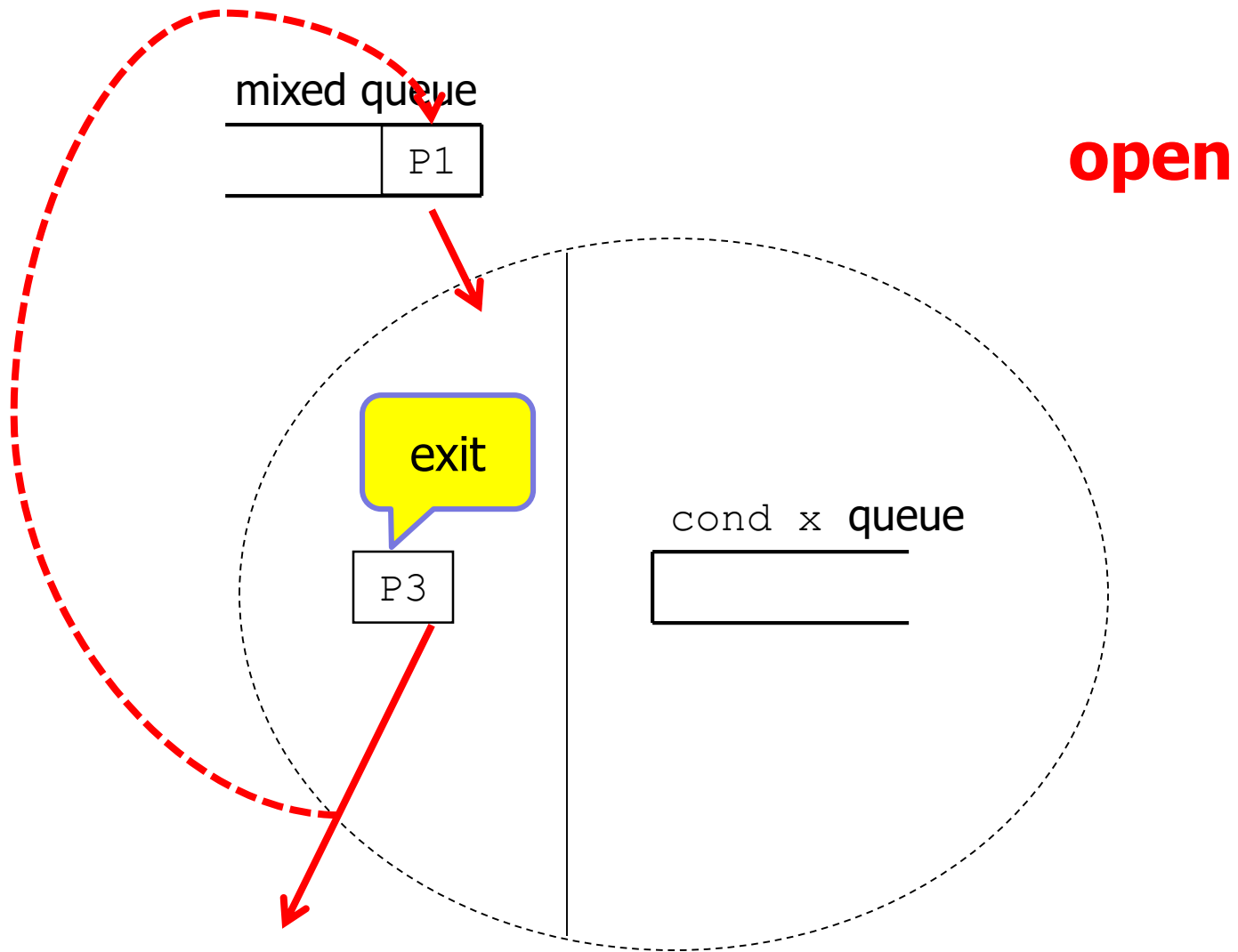


mixed queue



open

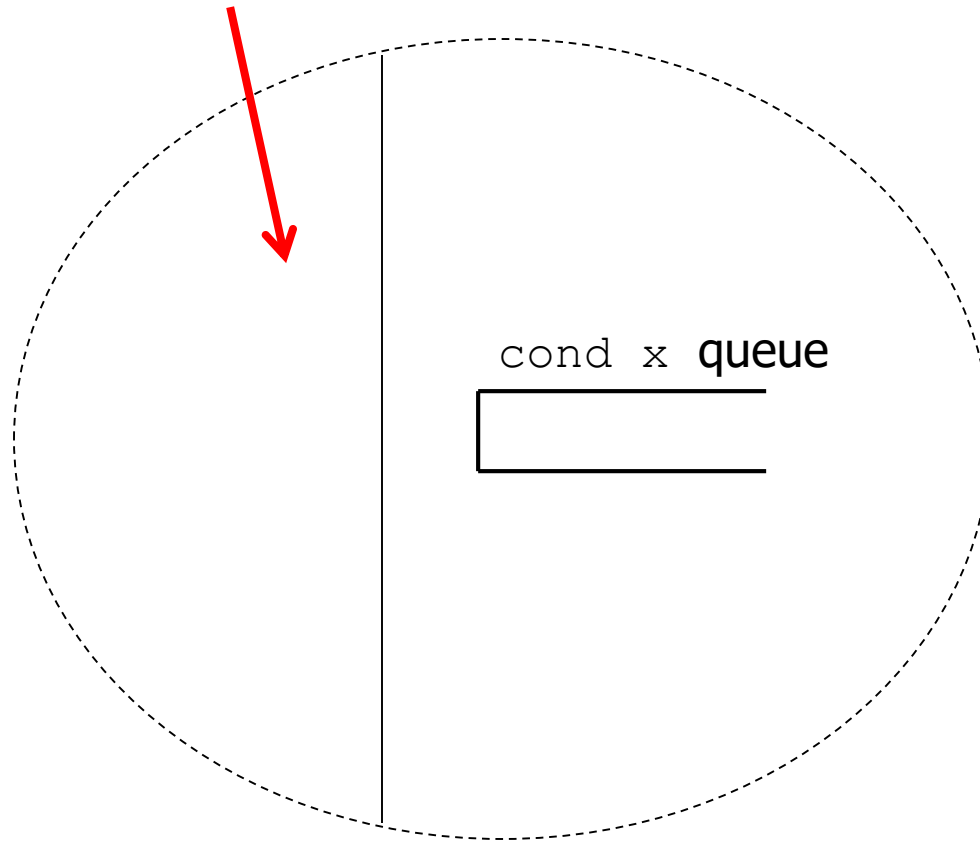




mixed queue



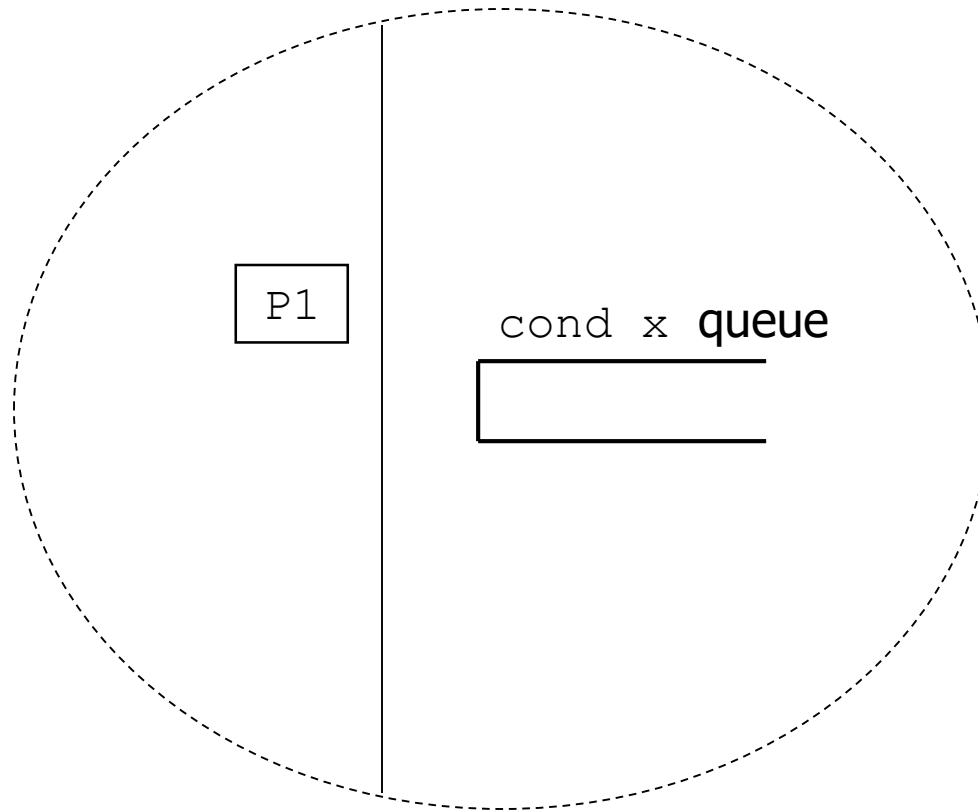
open



mixed queue



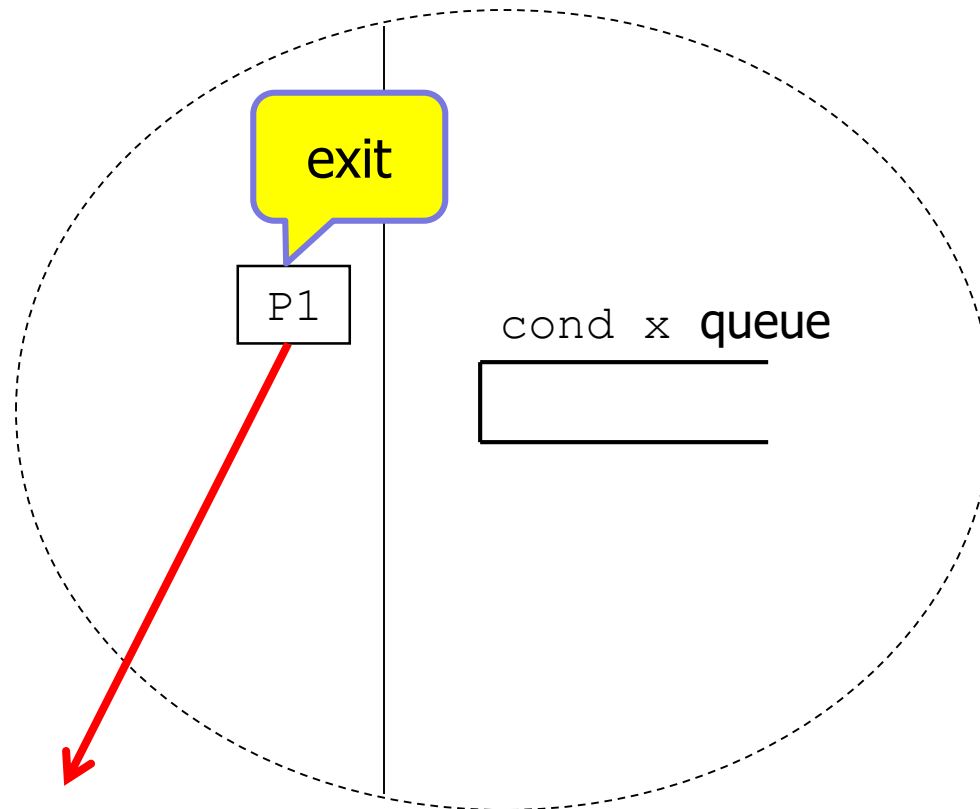
open



mixed queue



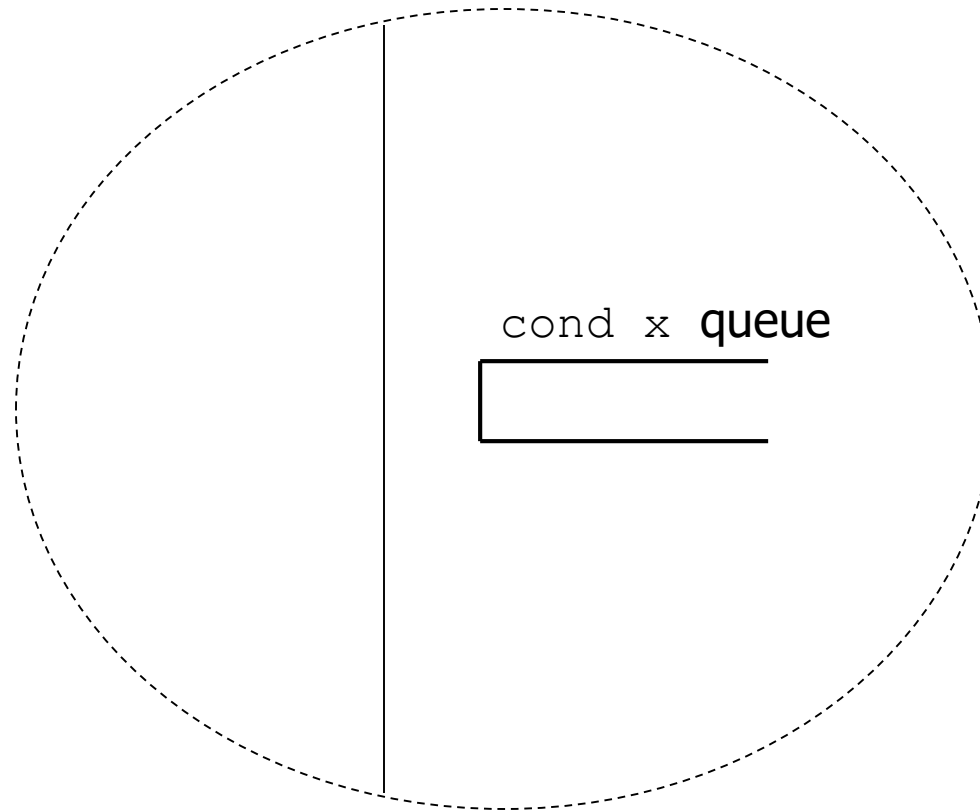
open



mixed queue



open



Συνθήκες ανταγωνισμού

- Ο μηχανισμός του ελεγκτή **λύνει** το πρόβλημα της συνθήκης ανταγωνισμού σε περίπτωση που ένα νήμα επιθυμεί να μπλοκάρει μέσα στον ελεγκτή
 - δεν μπορεί να παρεμβληθεί ανταγωνιστικό νήμα ανάμεσα στον έλεγχο της συνθήκης αναμονής και την κλήση της `wait`
- Όμως, μπορεί να προκύψει ένα **διαφορετικό είδος συνθήκης ανταγωνισμού**
 - ανάλογα με το μοντέλο προτεραιότητας του ελεγκτή
- Το eggshell/signal-block είναι ο πιο δίκαιος και διαισθητικά λογικός συνδυασμός (σειρά FIFO)

Έλεγχος προδιαγραφών

- Αρκετές γλώσσες και περιβάλλοντα εκτέλεσης υποστηρίζουν μηχανισμούς συγχρονισμού που έχουν πολλές ομοιότητες με τον μηχανισμό του ελεγκτή
- Βεβαιωθείτε ότι γνωρίζετε **ποιο μοντέλο** υλοποιούν
- Οι διαφορές ανάμεσα σε eggshell/open σε συνδυασμό με signal-block/signal-continue **επηρεάζουν** την ορθότητα του κώδικα
- Αν δεν υπάρχουν ξεκάθαρες προδιαγραφές (πράγμα σύνηθες), ο κώδικας πρέπει να σχεδιαστεί έτσι ώστε να λειτουργεί σωστά για όλους τους συνδυασμούς

Οι ελεγκτές δεν είναι πανάκεια

- Όπως κάθε εργαλείο/μηχανισμός συγχρονισμού, χρειάζεται προσοχή για να αποφεύγονται λάθη
- Πολλές φορές, τα λάθη / bugs **δεν** είναι προφανή
- Κλασικό πρόβλημα: **αδιέξοδα** (deadlocks)
 - ιδίως σε αντικειμενοστραφή περιβάλλοντα (όπου η ροή εκτέλεσης δεν είναι εντελώς προφανής εκ των προτέρων)
- **Nesting problem:** Σύνθεση πολύπλοκων τμημάτων λογισμικού από πιο απλά τμήματα/συστατικά
- **Reentrancy problem:** Επαναφορά, με έμμεσο τρόπο, πίσω στο ίδιο πλαίσιο συγχρονισμού

