Ett Småkryps Liv

Henrik Sommerland, Aleksander Lundqvist, Oliver Eriksson, Ludvig Strömberg, Oscar Wallster och Edvin Wahlberg

Introduktion

- Är demokrati och diktatorsstyre verkligen “All that it’s worked up to be”?
- Alternativet: Swarm Intelligence

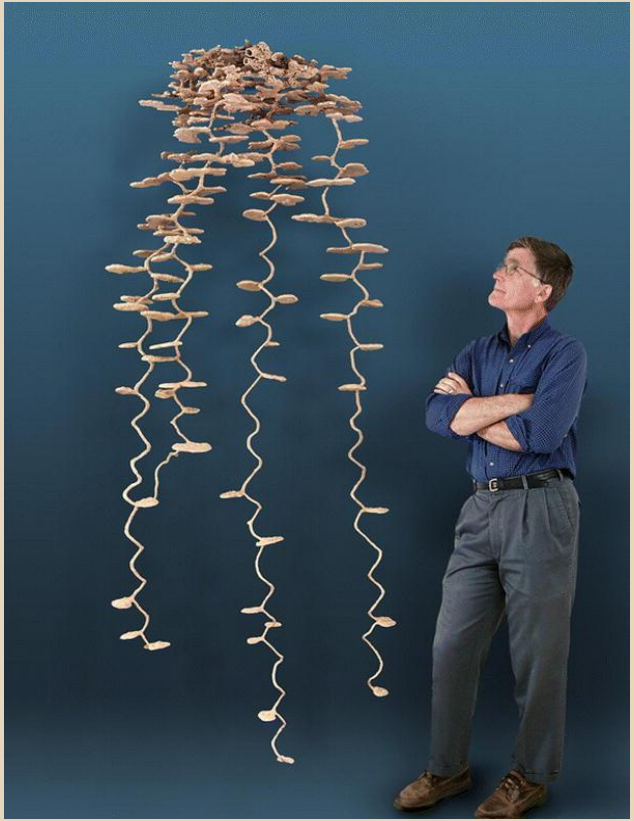
Introduktion

- Inspirationen var en National Geographic dokumentär
- Drottningen, är hon verkligen spindeln i nätet?
- ACO(Ant colonoy optimization) kan användas för att lösa Travelling Salesman Problem
- Erlangs Actor Model perfekt för en sådan simulering

Inspiration

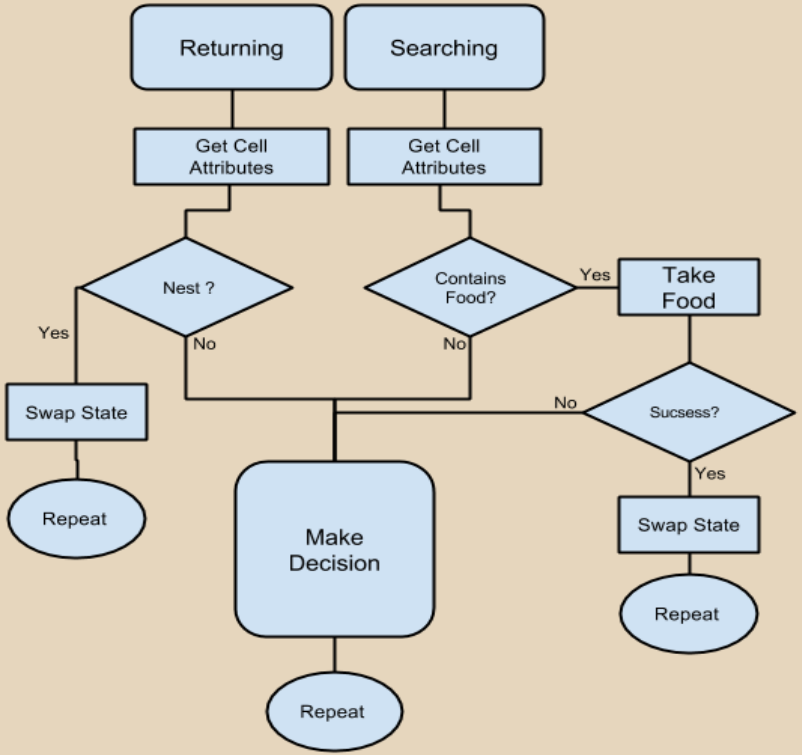
Svärmintelligens - myror

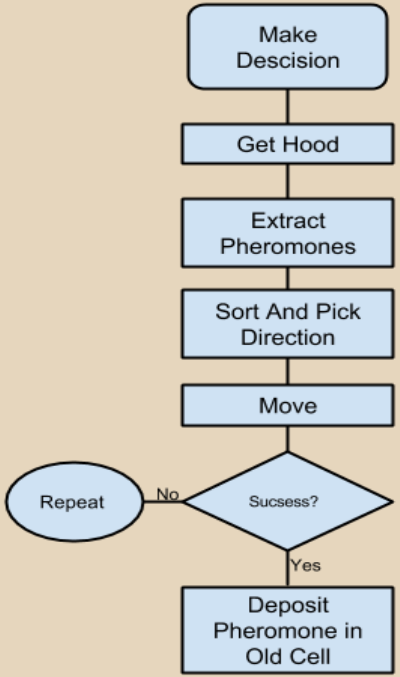
Individuellt värdelösa
Kollektivt produktiva



Demo

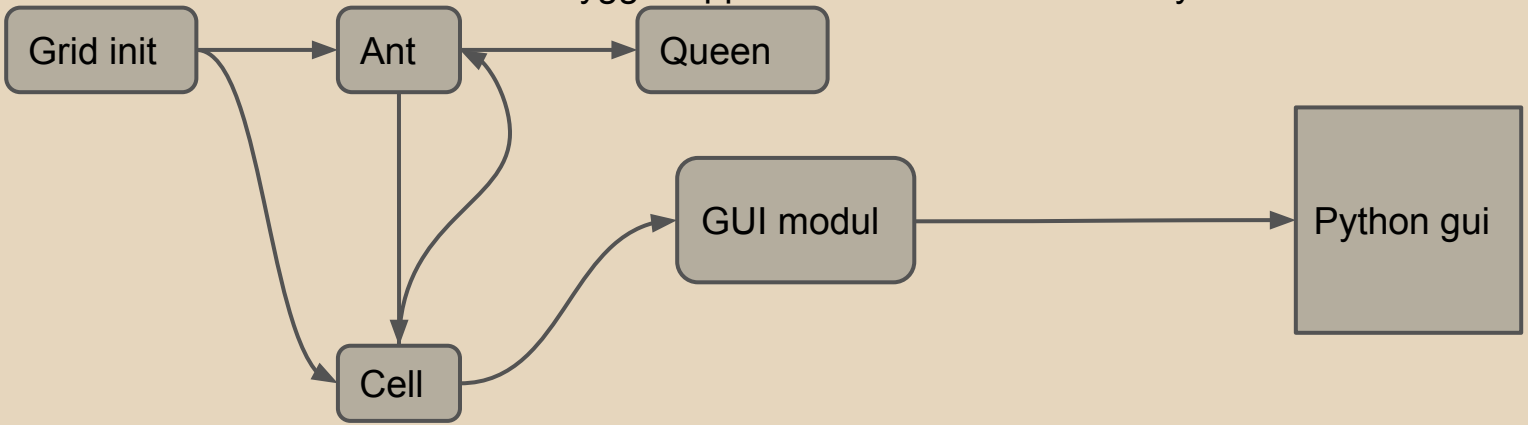
Algoritmen


Returning
Searching
Get Cell Attributes
Get Cell Attributes
Nest ?
Contains Food?
Yes
Take Food
Yes
No
No
Swap State
No
Sucsess?
Repeat
Make Decision
Yes
Swap State
Repeat
Repeat


Make Descision
Get Hood
Extract Pheromones
Sort And Pick Direction
Move
Repeat
No
Sucsess?
Yes
Deposit Pheromone in Old Cell

Systemarkitektur

- Kommunikation myra->cell (är det tomt, finns det feromoner, kan jag flytta mig hit, vad finns runtom)
- Kommunikation cell->myra (svarar)
- Cell->GUI modul (updates) sparar information och skicka till python för grafik
- Grid Init modulen bygger upp världen och startar alla myror.

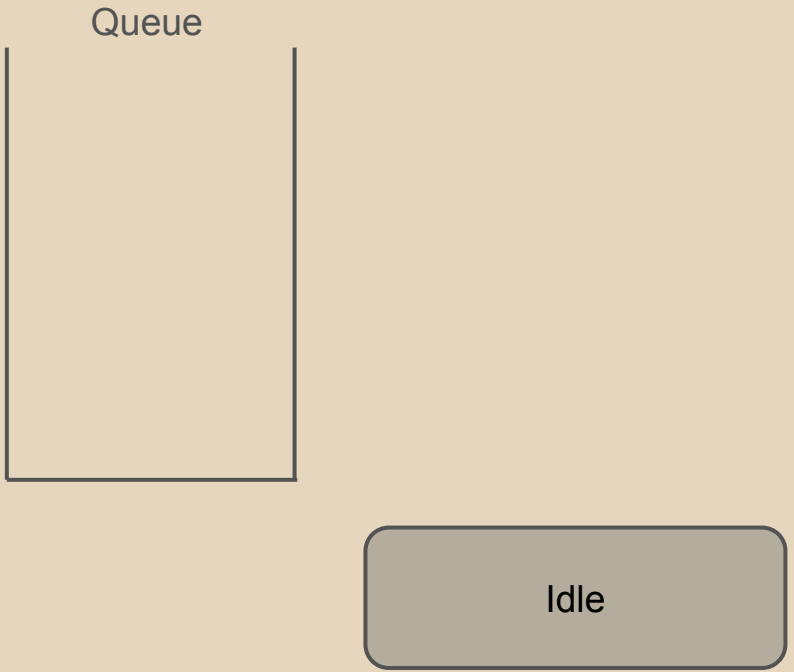


Concurrency

Event-Driven Programming med hjälp av actor modelen.

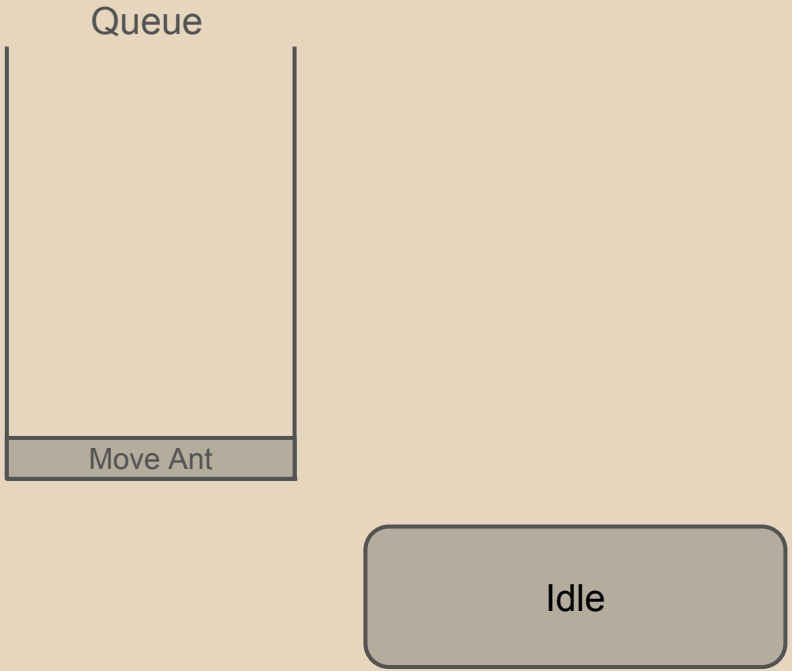
FIFO, Run to Completion

Låt det senast inkomna meddelandet diktera vilket tillstånd man hamnar i.



FIFO, Run to Completion

Låt det senast inkomna meddelandet diktera vilket tillstånd man hamnar i.



FIFO, Run to Completion

Låt det senast inkomna meddelandet diktera vilket tillstånd man hamnar i.

FIFO, Run to Completion

Problem!

Hur hanterar vi ett “Set State” Medelande medans vi väntar på ett svar från en annan process?

Vi kan kanske se om “Set State” meddelandet inte vill ändra på något som är beroende utav huruvidare vi har en myra i cellen eller vi kanske låter “Set State” tvinga fram en roll back och på så vis negerar “Move Ant” begäran och sedan skickar ett medelande till cellen dit myran flyttade som sägger åt den att om myran kunde gå dit så skall den återställa sig till det tillståndet som den var i tidigare och sedan låter vi cellen vänta på ett svars medelande om att cellen myran skulle gå till har genomfört en korrekt rollback.

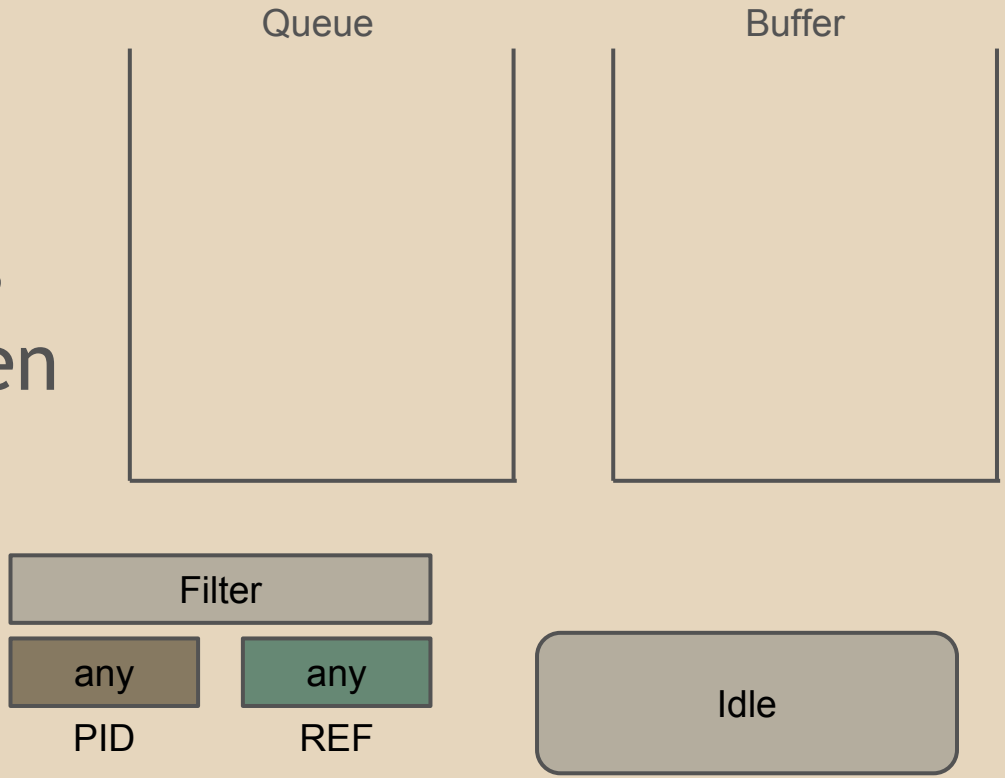
Await Place Reply

“State-driven Message Handling”

Låt processens tillstånd diktera vilka medelanden som accepteras!

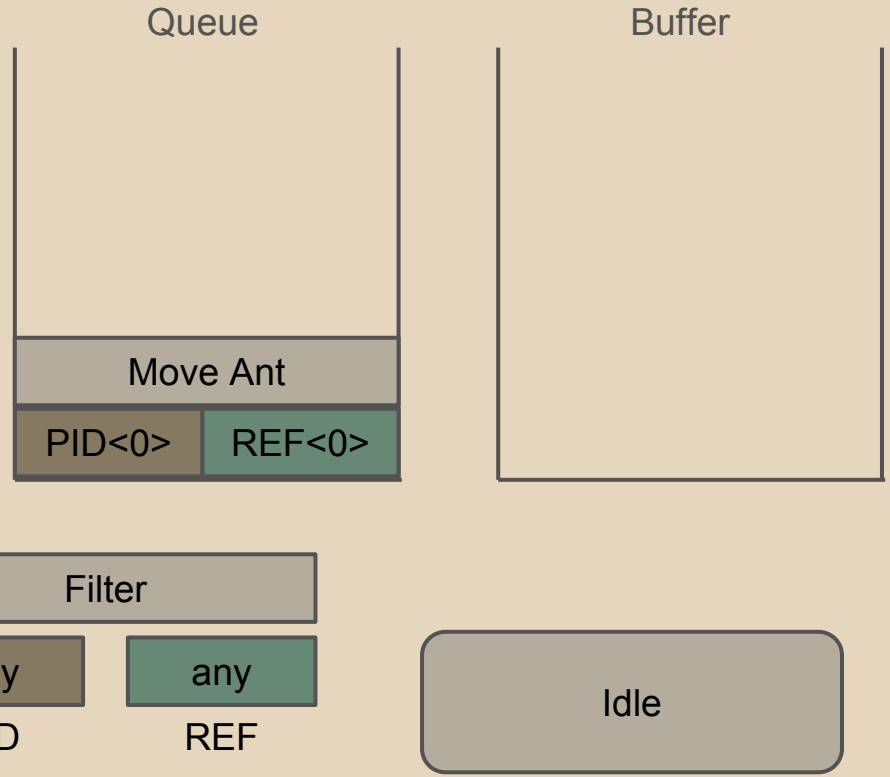
“State-driven Message Handling”

Vi introducerar en medelande buffer så att vi kan blocka tills vi får rätt medelanden utan att kasta bort medelande kön.



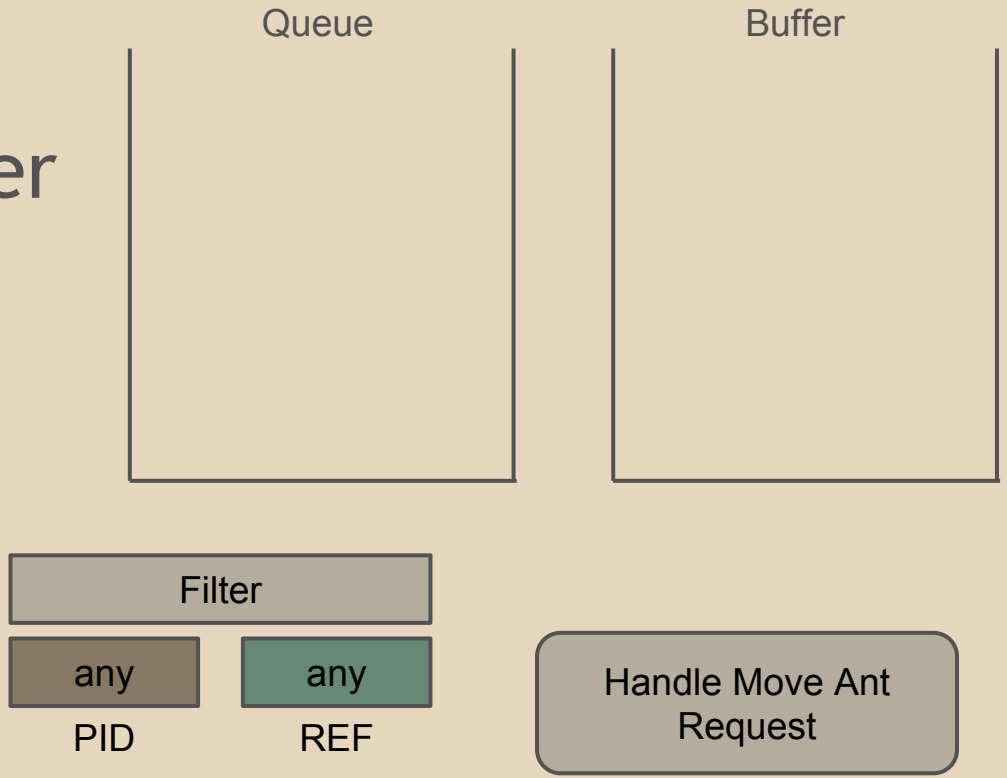
“State-driven Message Handling”

Vi introducerar en medelande buffer så att vi kan blocka tills vi får rätt medelanden utan att kasta bort medelande kön.



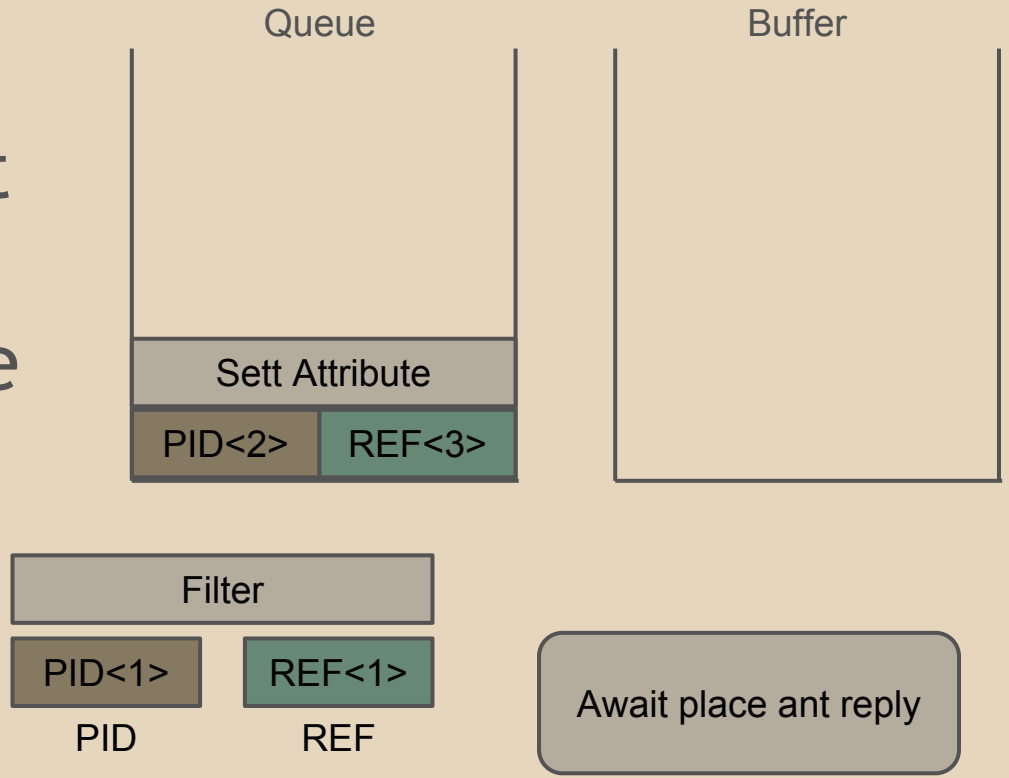
“State-driven Message Handling”

Processen hanterar medelandet och väljer att skicka ett place ant request till PID <1> med refference REF<1>.



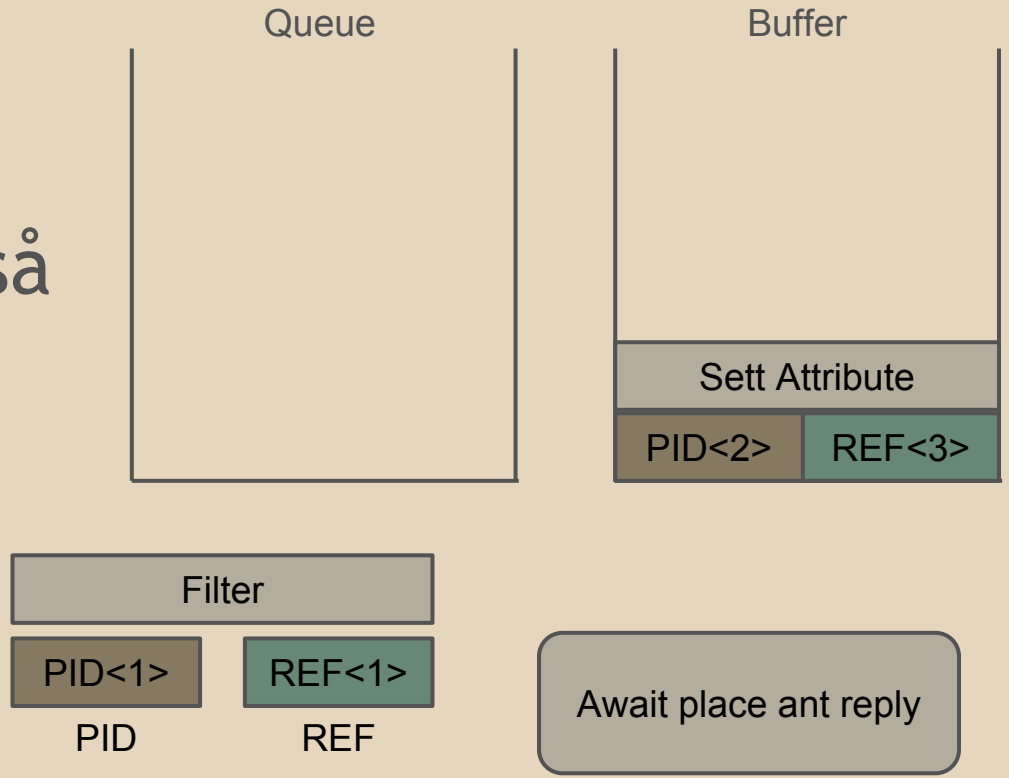
“State-driven Message Handling”

Processen sätter sitt message filter till att bara filtrera alla medelanden som inte kommer från PID<1> och har REF<1> som referens.



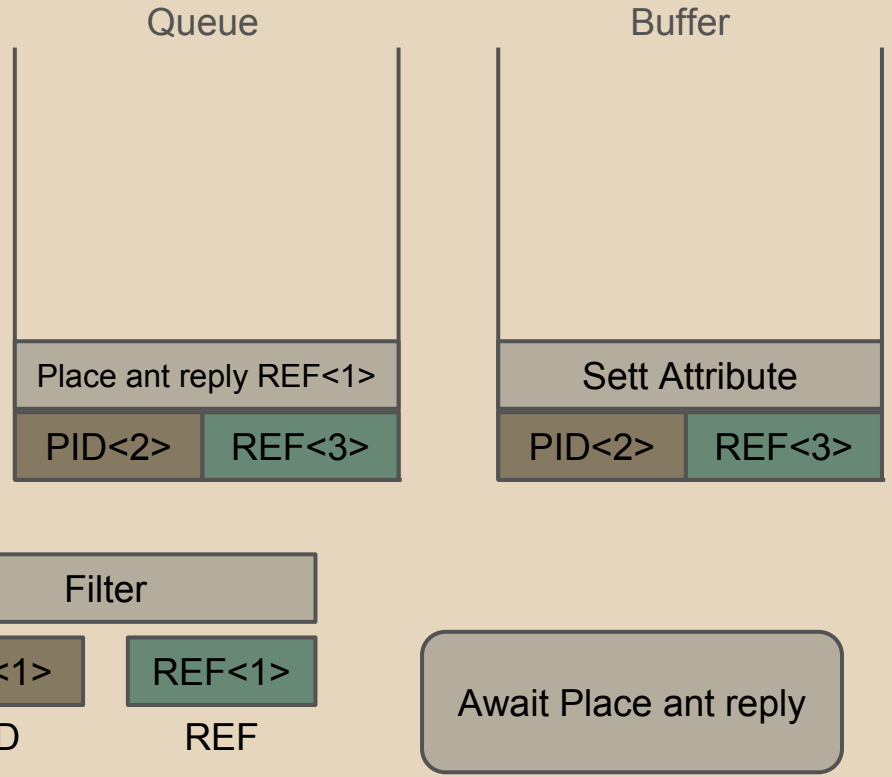
“State-driven Message Handling”

Om ett medelande kommer med fel Pid eller fel Refference så sparas detta madelande i en buffer.



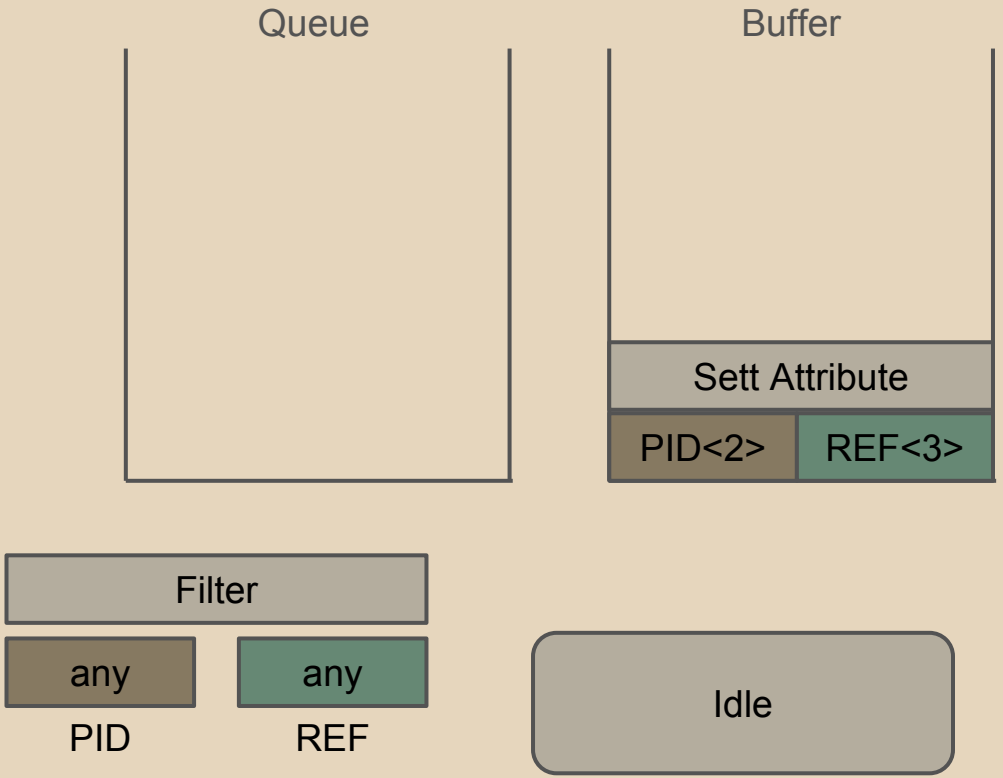
“State-driven Message Handling”

När rätt medelande kommer in så behandlas det och filtret återställs



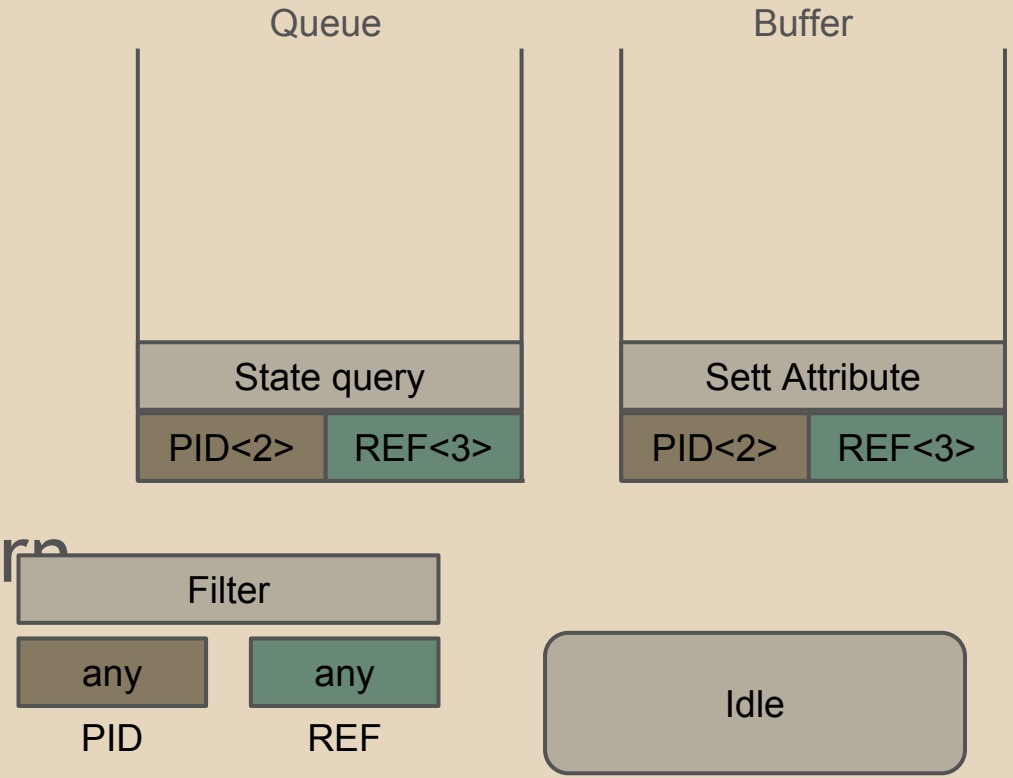
“State-driven Message Handling”

När rätt medelande kommer in så behandlas det och filtret återställs



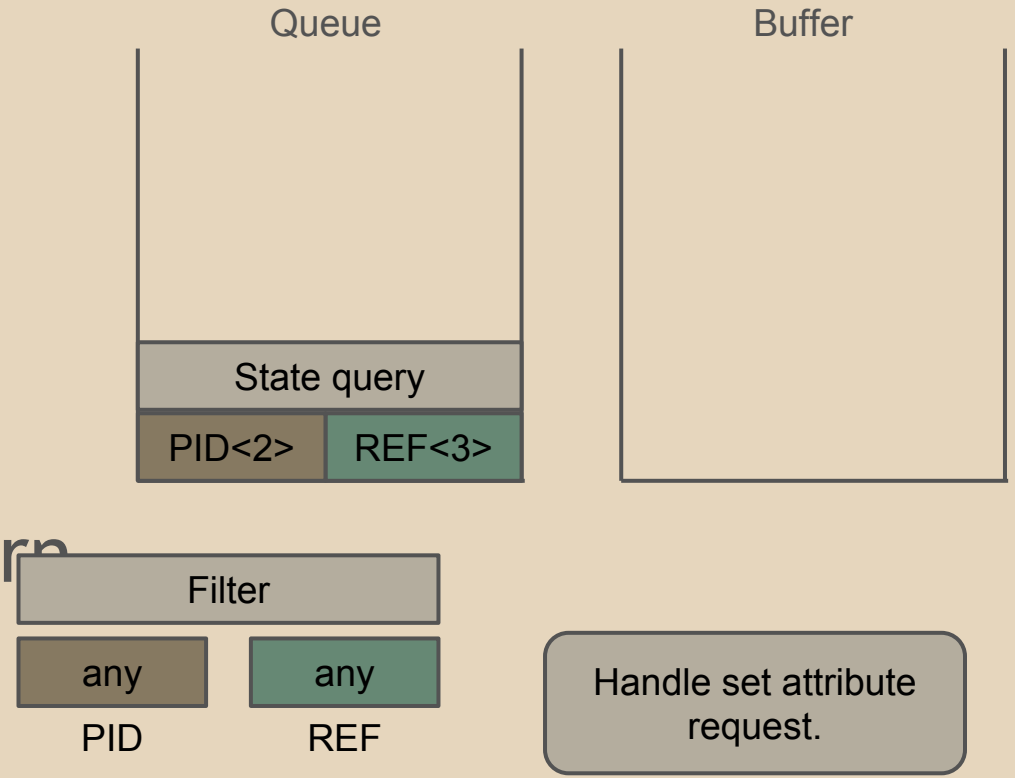
“State-driven Message Handling”

När processen sedan återgår till det tillstånd där den accepterar nya requests så tar den medelandena i buffern först.



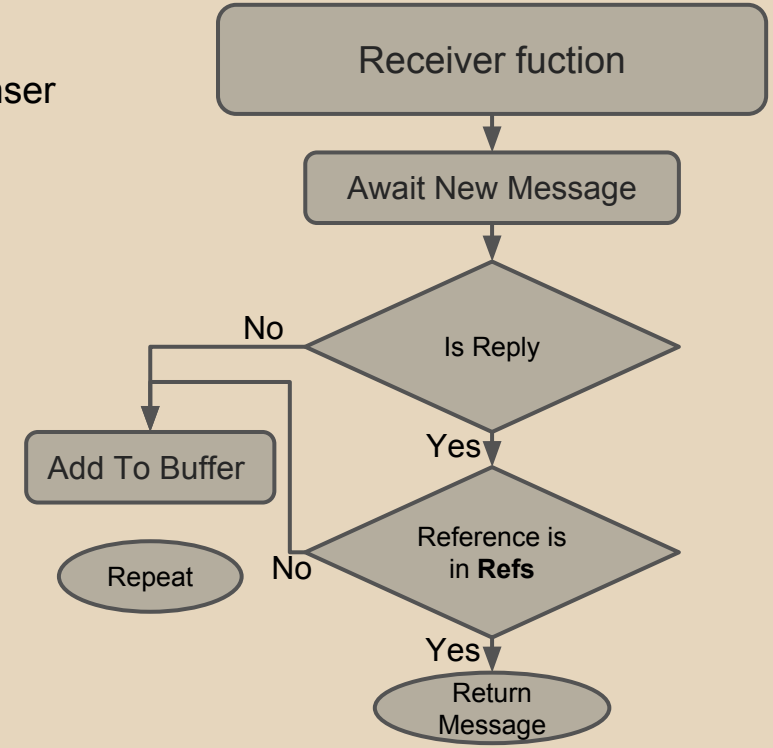
“State-driven Message Handling”

När processen sedan återgår till det tillstånd där den accepterar nya requests så tar den medelandena i buffern först.



Medelande filtrering.

Refs = Lista med medelande referenser



Deadlocks

Frågan för oss är inte huruvida deadlocks kommer att uppstå utan hur vi detekterar deadlocks och hanterar dem.

Detection And Prevention

Vi kan med några enklare predikat konstruera en metod som kan:

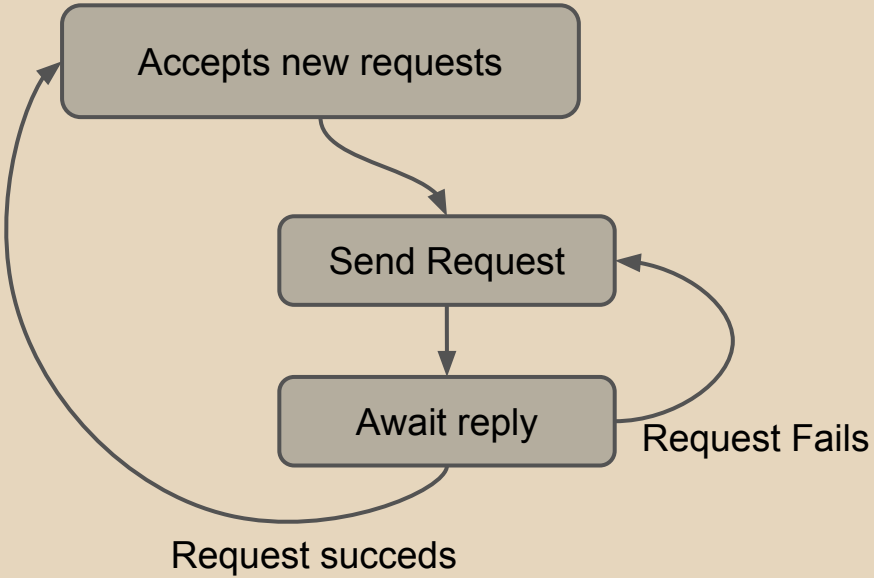
- Upptäcka enklare deadlocks.
- Uppskatta/Gissa att deadlocks har uppstått.
- Lösa “alla” deadlocks.

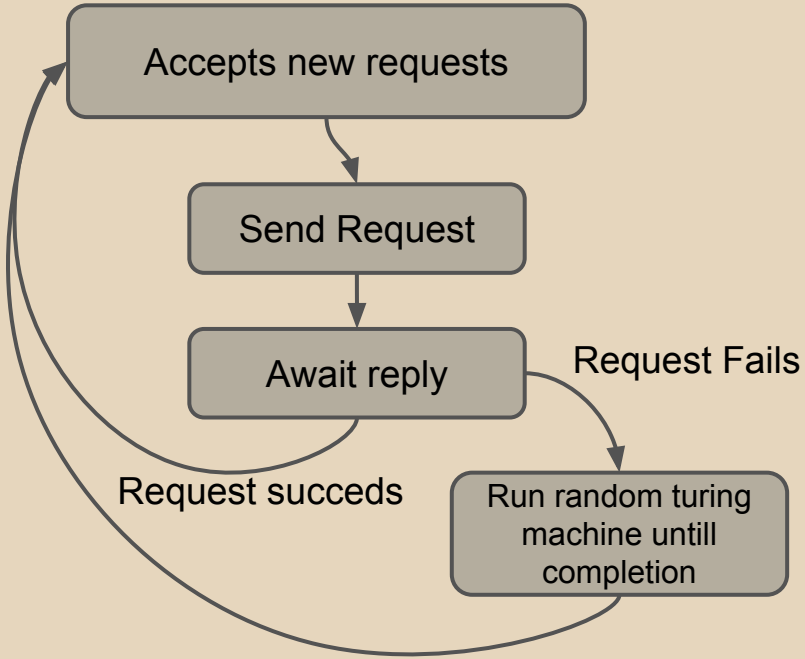
Detection And Prevention

Dock så kräver denna metod att vissa krav uppfylls:

- En begäran kan misslyckas och betendet vid en misslyckad begäran är alltid väldefinierat.
- Alla processer kommer invänta ett svar efter att en förfrågan har skickats.
- **Från det att ett svar på en förfrågan har inkommit kommer processen alltid att inom en finit tid drivas till ett tillstånd där den accepterar inkommande förfrågningar**

Otillåtna konfigurationer



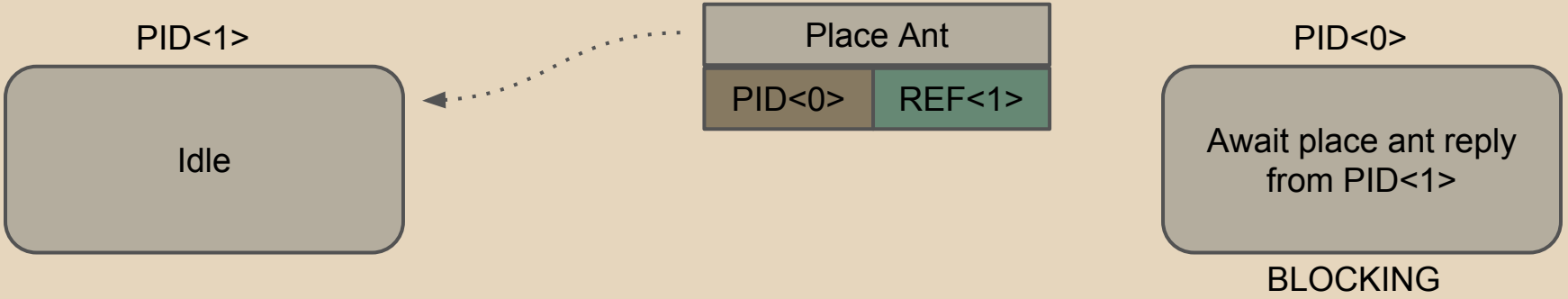


Detection And Prevention

Hur upptäcker vi enkla deadlocks?

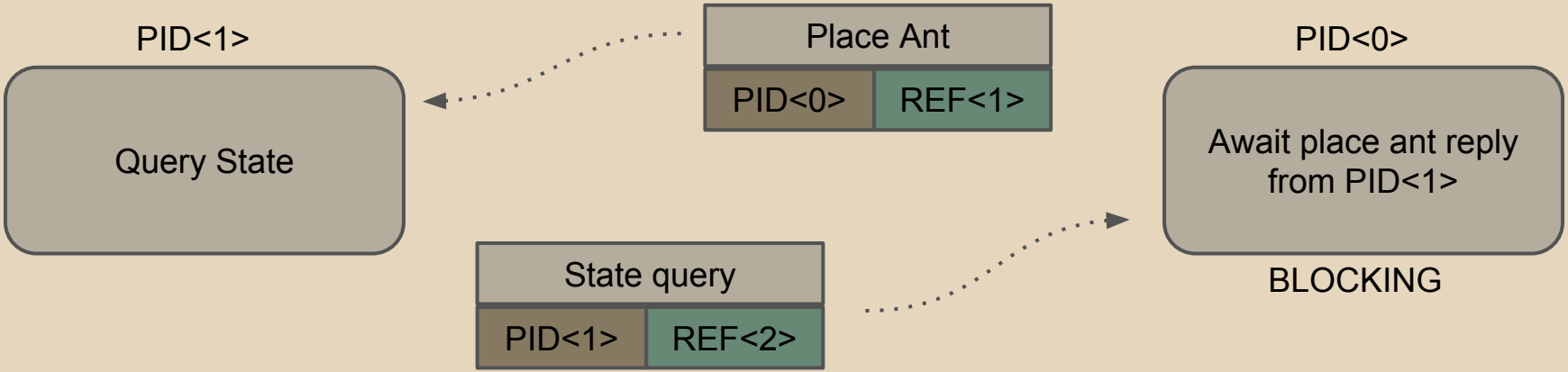
Detection And Prevention

En process skickar ett medelande till en annan process och väntar på svar.



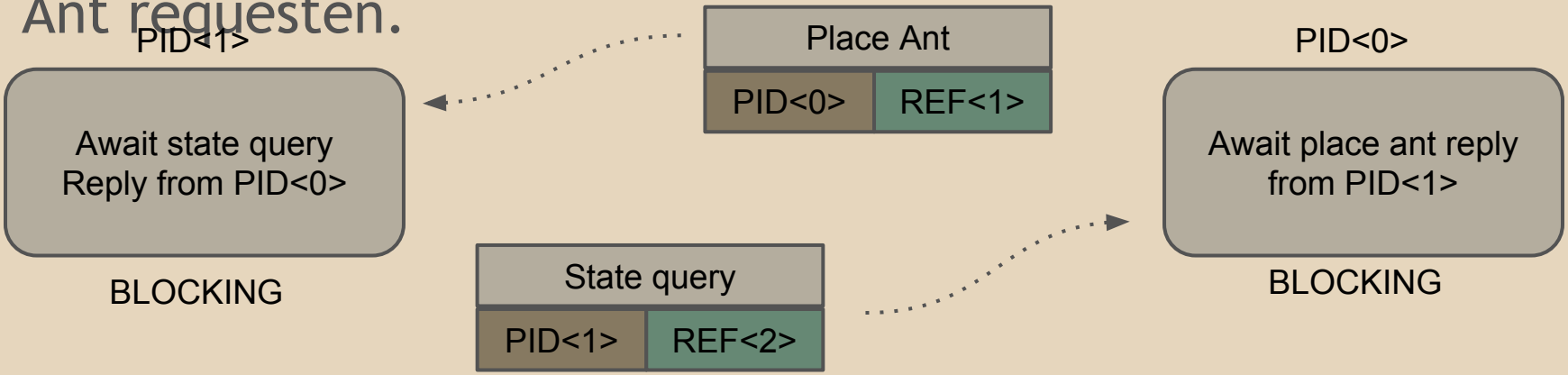
Detection And Prevention

Medans processen väntar på svar så får cellen en request från processen den väntar på svar från.



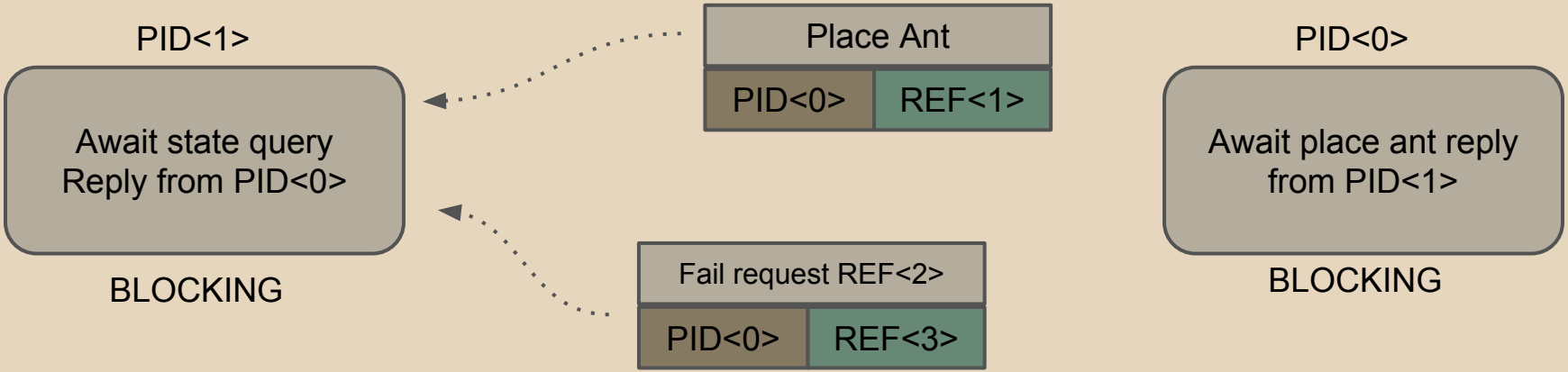
Detection And Prevention

Ett deadlock har uppstått! PID<1> blockerar och väntar på svaret från sin request och kommer inte att hantera Place Ant requesten.



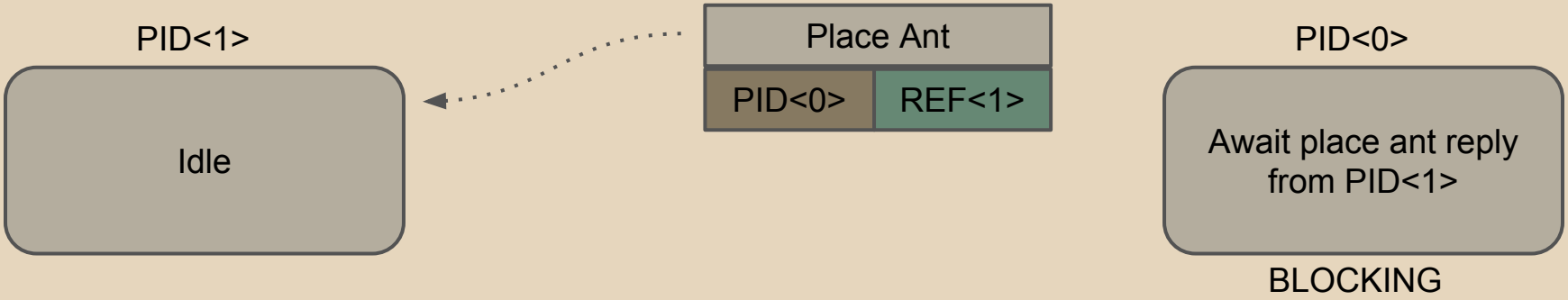
Detection And Prevention

Processen som uptäckte deadlocket skickar automatiskt ett svar på den konflikterande requesten.



Detection And Prevention

Processen som fick fail replyet kommer enligt våra predikat att drivas tillbaka till det tilståndet då det accepterar nya requests.



DEADLOCK LÖST!

PID<1>

Processing state querry

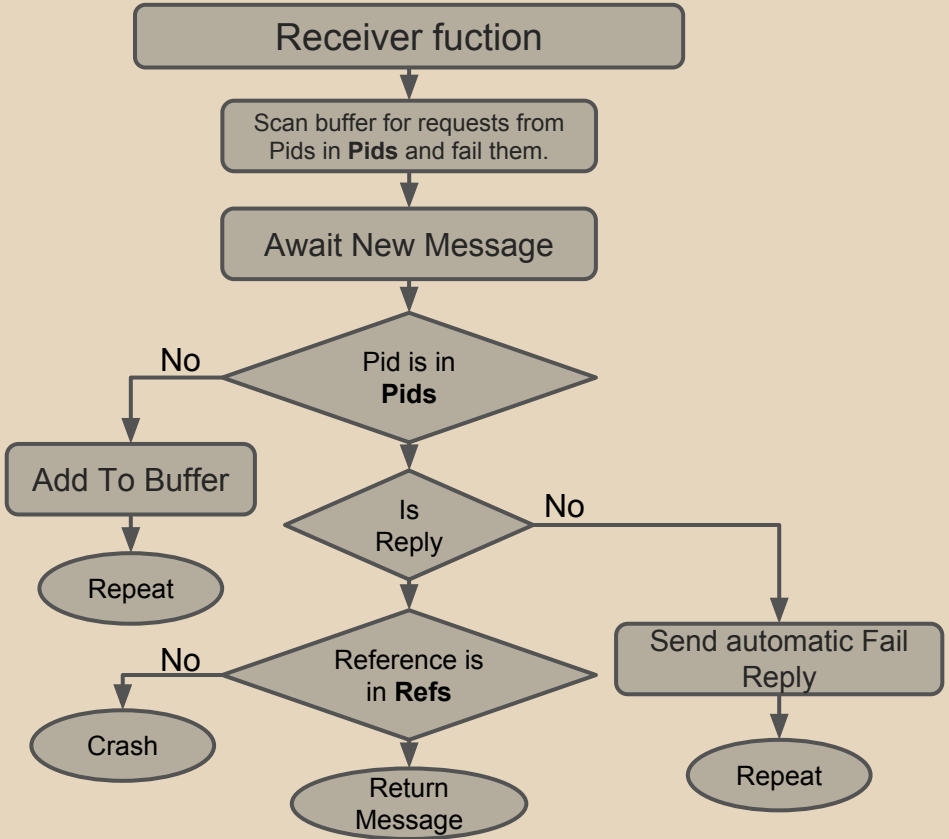
PID<0>

Await place ant reply from PID<1>

BLOCKING

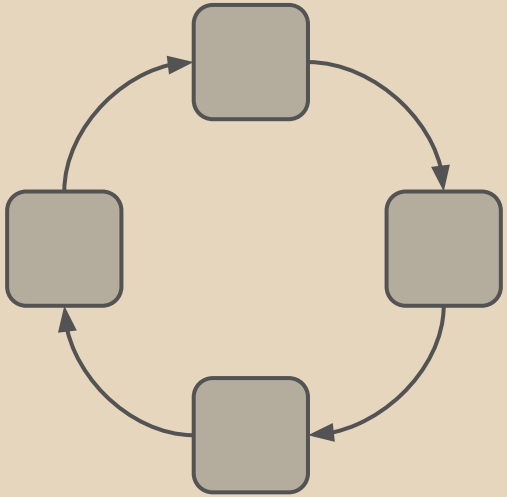
Löser Enkla Deadlocks

Refs = Lista med medelande referenser.
Pids = Lista med mottagarnas PIDs



Komplicerade Deadlocks

Vad händer ifall vi har en längre kjedja utav beroenden som skapar ett deadlock?



Komplicerade Deadlocks

Vi förutsätter att om en process inte har fått svar på en request inom en viss tid så antar vi att ett deadlock har uppstått.

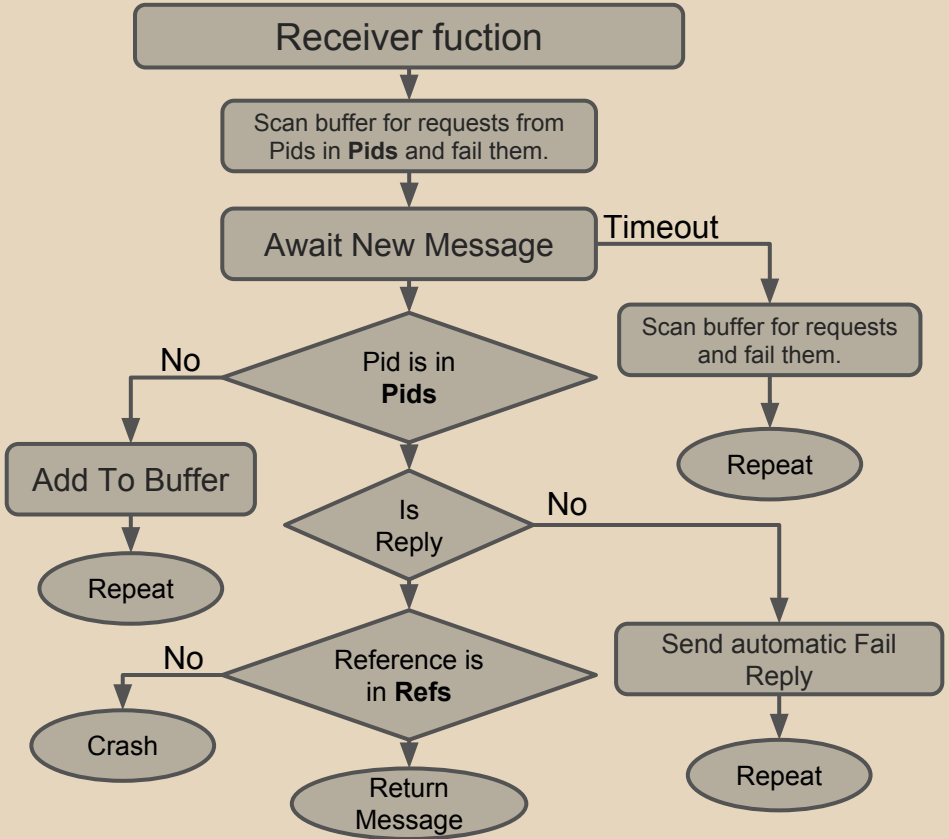
Processen kommer då att automatiskt skicka “fail” replys till alla requests som finns i medelande buffern.

Komplicerade Deadlocks

Detta leder till att alla requests kommer att besvaras inom en finit tid även ifall ett deadlock har uppstår. Detta leder till att inga deadlocks kan ligga kvar längre än den timeout som har definierats.

Löser Alla Deadlocks

Refs = Lista med medelande referenser.
Pids = Lista med mottagarnas PIDs



Fördelar

- Trivialt att implementera. (~150 rader med erlang)
- Försummbar overhead
  Inget behov utav att analysera hela systemet.
  Försumbar om delenade buffern relativt är liten.
- Inga rollbacks

Nackdelar

- Upptäcker “falska” deadlocks
- Predikaten som metoden bygger på är inte applicerbara på alla system
- Timeout parametern måste bestämmas i förväg och måste finjustreras.

  En för lång timeout leder till att deadlocksen kommer att sprida ut sig och ligga kvar. Systemet kommer att “lagga”

  En för kort timeout kommer att leda till att falska deadlocks kommer att upptäckas förofta och requests kommer att failas för ofta.

Verktyg

- Git
- Slack
- Trello

Arbetsmetodik

- Möten
- Arbetsfördelning
- Sammankoppling

Guldkorn

- Medelande filtreringen
- Fritt från deadlocks!
- Python Erlang interface

Ruttna ägg

- Kod kvalitet
- Inga peer reviews
- Rörig kod i Ant modulen