Distribuované systémy a výpočty

X36DSV

Jan Janeček

Ukončení výpočtu

Dijkstra - Scholten

Proměnné - legenda

Defin – deficit na vstupu
Defout – deficit na výstupu
Others – seznam dalších žádajících procesů

```
begin                                          { inicializace }
   Defin:=0; Defout:=0; Others:=
end

receiving MESSAGE from j do                    { příjem žádosti aplikace }
  begin
    if DefIn=0
    then Parent := j
    else Others := Others+j;
    DefIn := DefIn+1
  end
```

Ukončení výpočtu

Dijkstra - Scholten

```
receiving SIGNAL from j do                    { příjem odpovědi aplikace }
  DefOut:=DefOut-1;

sending MESSAGE to j do                       { odeslání žádosti aplikace }
    { possible if DefIn>0 }
    DefOut := DefOut+1;

sending SIGNAL to (Oth=any of Others) do      { odeslání odpovědi aplikace }
    { possible if (DefIn>1 }
    begin
        Others := Others-Oth;
        DefIn := DefIn-1
    end

sending SIGNAL to Parent do                   { odeslání odpovědi aplikace }
    { possible if (DefIn=1 and DefOut=0) }
    DefIn := DefIn-1
```

Ukončení výpočtu

Dijkstra – Feijen – Van Gasteren

Proměnné - legenda
State – stav procesu
Color – „barva“ procesu
TPresent – informace o vlastnictví tokenu
TColor – barva přijatého tokenu

```
begin                                   { inicializace }
   TPresent := F; Color := WHITE
end

receiving MESSAGE do                    { příjem zprávy aplikace }
   State := ACTIVE

waiting MESSAGE or State=TERMINATED do
   State := PASSIVE                     { čekání na zprávu aplikace }

sending MESSAGE to j begin              { odeslání zprávy procesu s indexem j>i }
   if i<j then Color := BLACK
```

Ukončení výpočtu

Dijkstra – Feijen – Van Gasteren

```
when received TOKEN(ct) from i+1 do     { příjem zprávy TOKEN }
   begin
      TPresent := T;
      TColor := ct;
      if i=0 then
         if Color=WHITE and TColor=WHITE
         then { TERMINATION DETECTED }
         else TColor := WHITE
   end

when TPresent and State=PASSIVE do
   begin                                { předání zprávy TOKEN následníkovi }
      if Color=BLACK then TColor := BLACK;
      TPresent := F;
      send TOKEN(TColor) to i-1;
      Color := WHITE
   end
```

Ukončení výpočtu

Misra

```
when received MESSAGE do          { příjem zprávy aplikace }
   begin
      State := ACTIVE;
      Color := BLACK
   end

when waiting MESSAGE do           { čekání na zprávu aplikace }
   State := PASSIVE

when received TOKEN(j) do         { příjem zprávy TOKEN }
   begin
      nb := j;
      TPresent := T;
      if nb=Size(C) and Color=WHITE then
         { TERMINATION DETECTED }
   end
```

Ukončení výpočtu

Misra

```
when TPresent and State=PASSIVE        { odeslání zprávy TOKEN }
   begin
      if Color=BLACK then nb := 0
      else nb := nb+1;
      send TOKEN(nb) to Succesor(C,i);
      Color := WHITE;
      TPresent := F
   end

begin                                  { inicializace }
   Color := BLACK; TPresent := F; nb := 0
end
```

Ukončení výpočtu

Misra

```
when TPresent and State=PASSIVE        { odeslání zprávy TOKEN }
   begin
      if Color=BLACK then nb := 0
      else nb := nb+1;
      send TOKEN(nb) to Succesor(C,i);
      Color := WHITE;
      TPresent := F
   end

begin                                  { inicializace }
   Color := BLACK; TPresent := F; nb := 0
end
```

Ukončení výpočtu

Rana

```
when received MESSAGE(m) do
    begin
        hreset;
        LocPred:=F
    end

when change to T of LocPred do
    begin
        LastTime:=Clock;
        send DETECTION(LastTime,1) to successos
    end
```

Ukončení výpočtu

Rana

```
when received DETECTION(time,number) do
    begin
        hreset;
        if number=n and LocPred then
            begin
                Term:=Y;
                send TERMINATED to successor
            end;
        if number<>n and LocPred then
            begin
                if Time>=LastTime then
                    send DETECTION(Time,Number+1) to successor
            end
    end
```

Ukončení výpočtu

Rana

```
when received TERMINATED do
    if Term=N then
        begin
            Term:=Y;
            send TERMINATED to successor
        end
```

Replikace

Performance enhancement

- concurrent read-only accesses
- sequential updates

Enhanced availability

$1 - p = 1 - p^n$

Fault tolerance

$t < n/2$ - fail stops

$b < n/3$ - Byzantine failures

Quora

Majority quorum

quorum size = (n+1)/2

Maekawa's quorum

quorum size = $\sqrt{n}$

Tree quorum

quorum size = $\sqrt{n}$ . . . (n+1)/2

Quora

set of nodes/sites

$$S = \{ s_1, s_2, \ldots, s_n \}$$

coterie

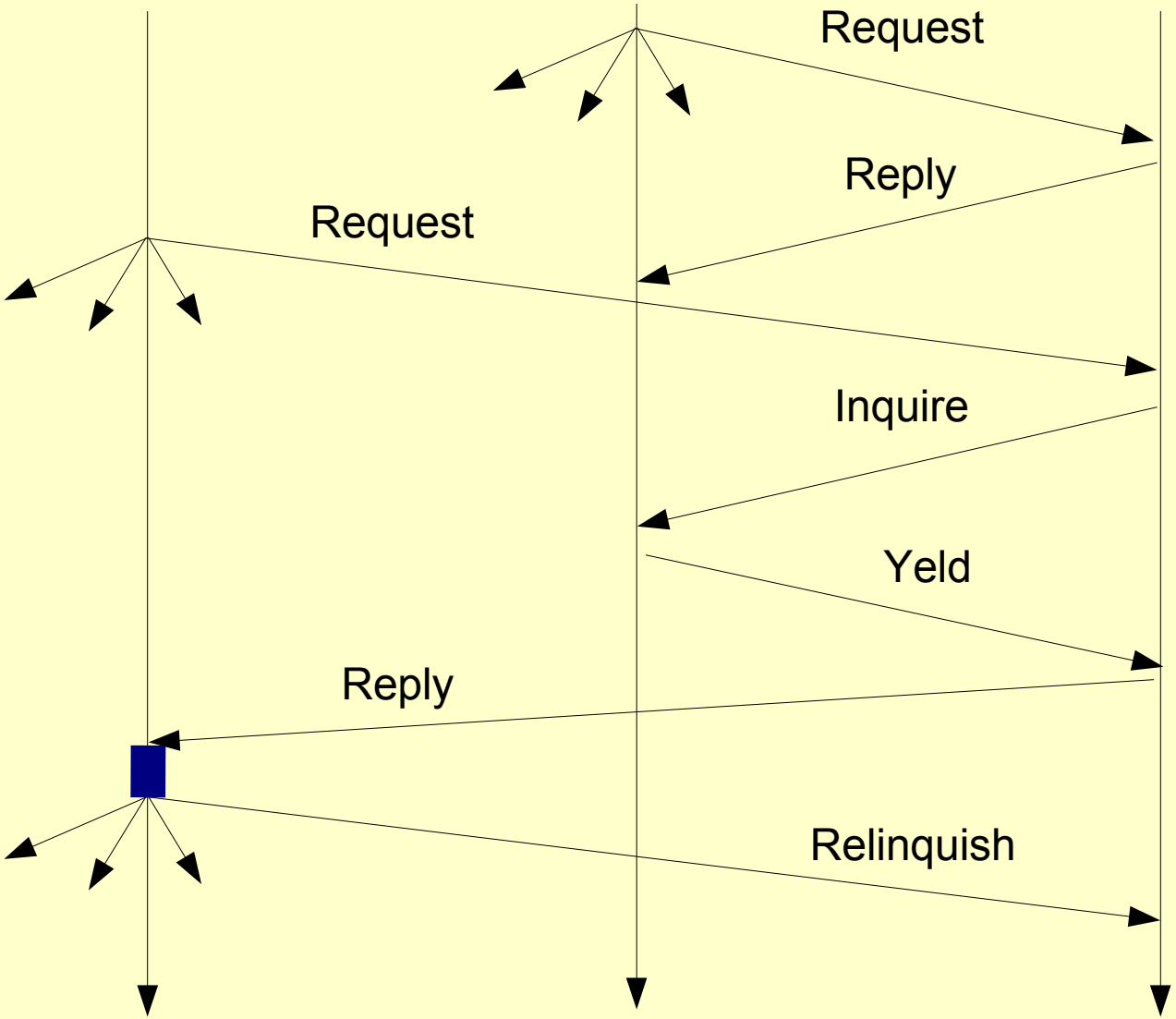
$$C = \{ Q_1, Q_2, \ldots, Q_n \}, \quad Q_i \neq \emptyset \wedge Q_i \subseteq S$$

quorum

$$Q_i \cap Q_j \neq \emptyset, \; Q_i, Q_j \in C, \; i \neq j$$ - intersection property

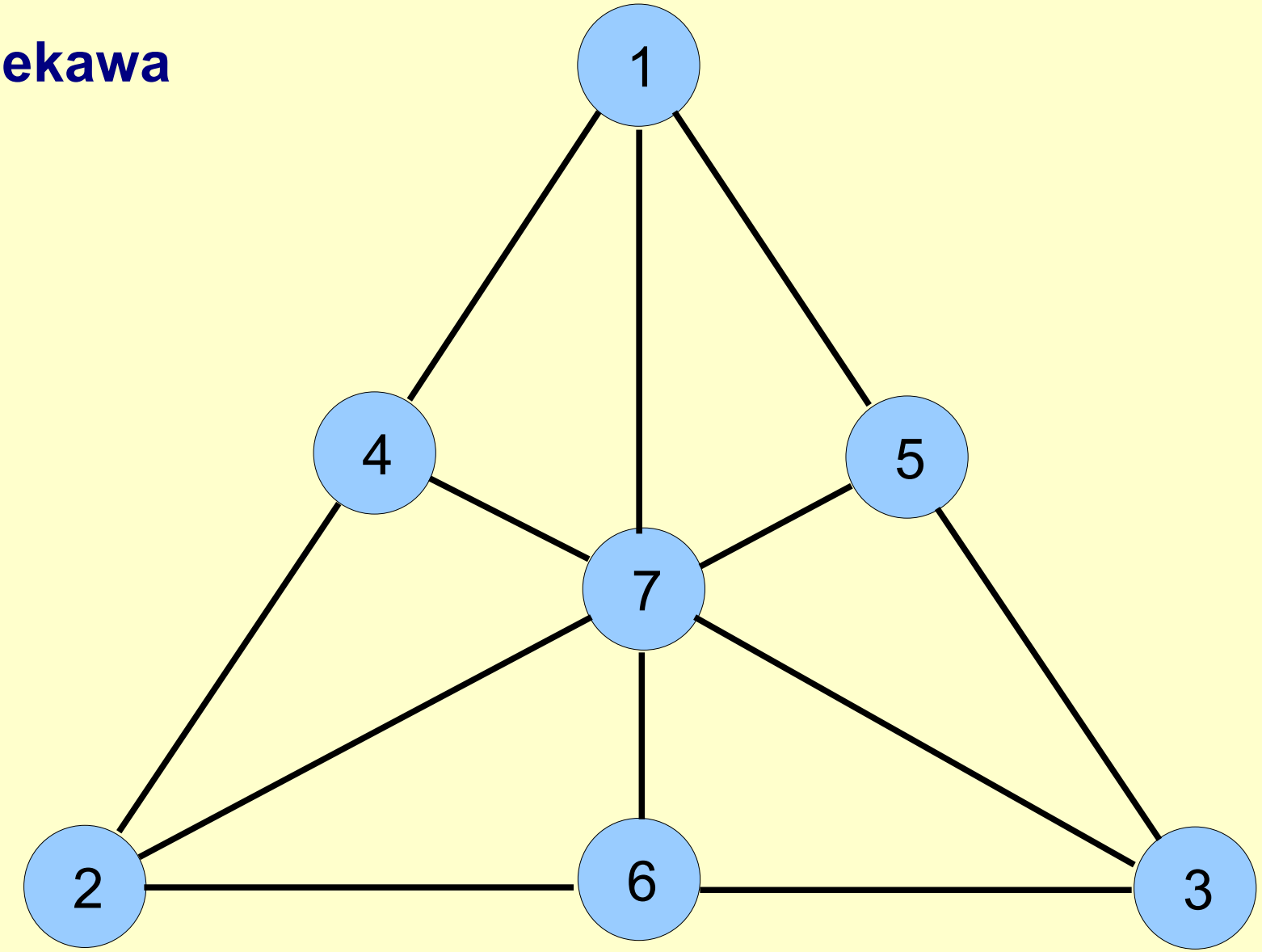
$$Q_i \not\subset Q_j, \quad Q_i, Q_j \in C, \; i \neq j$$ - minimality property

Quora



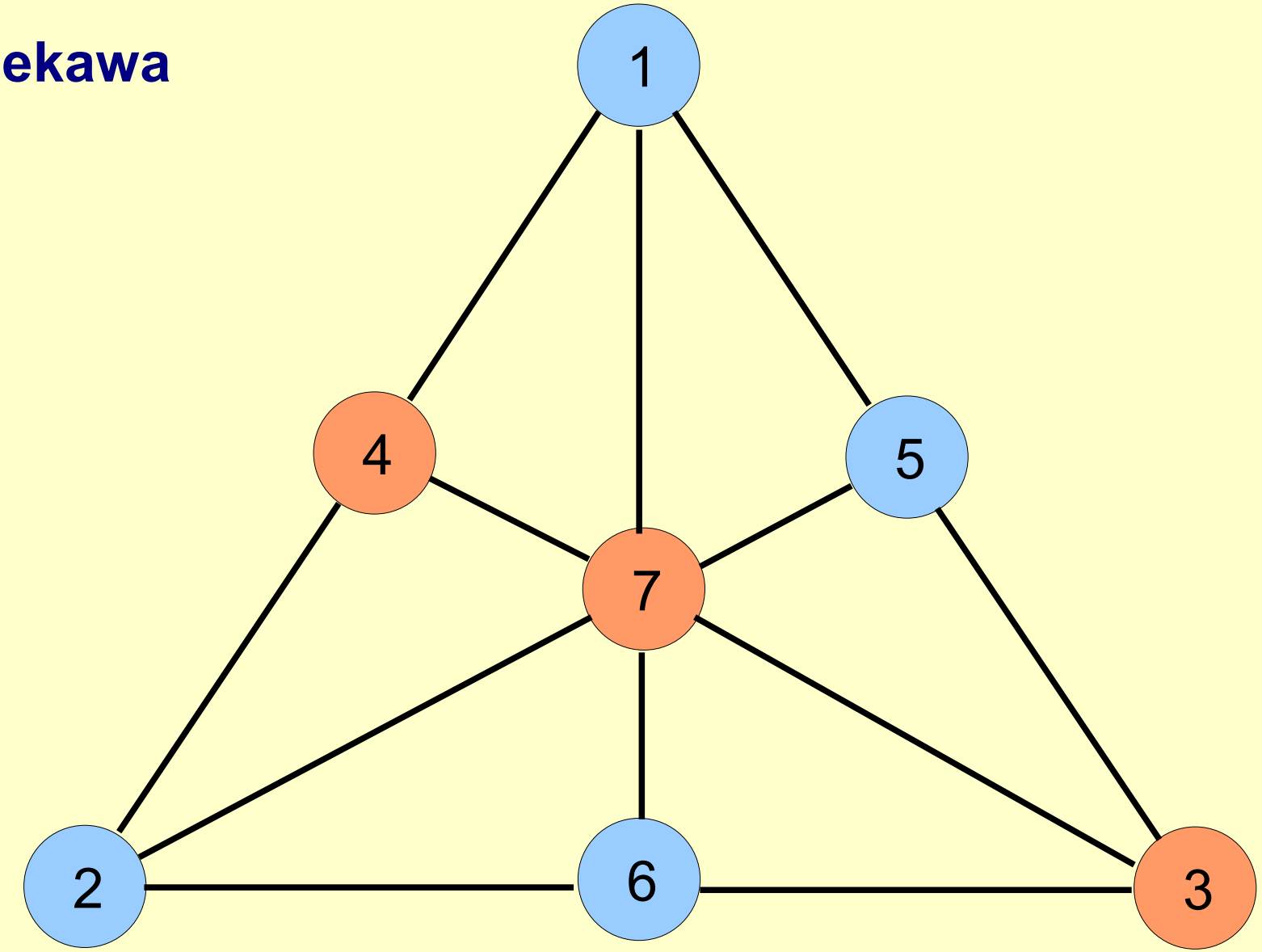
Quora

Maekawa



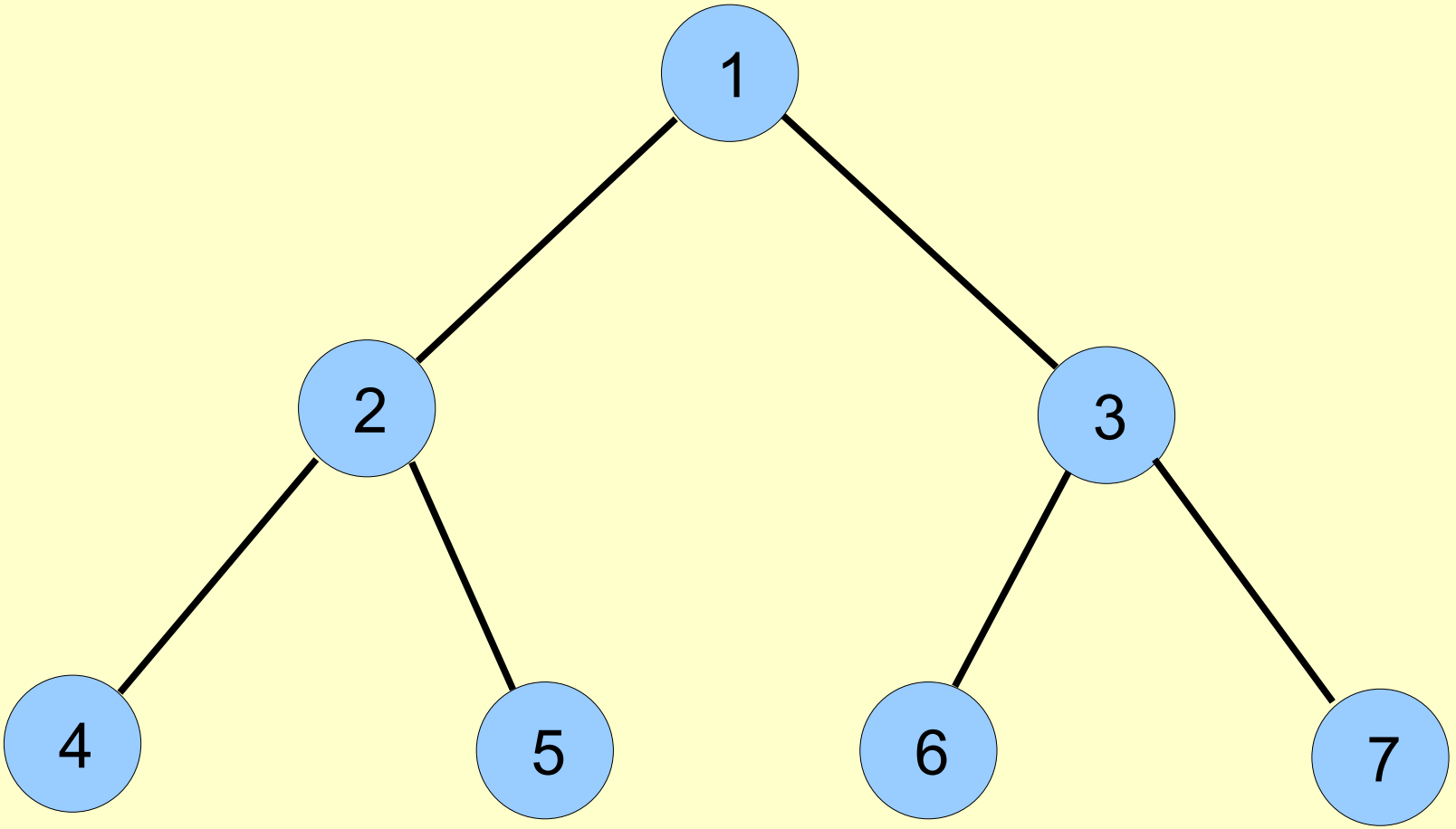
Quora

Maekawa



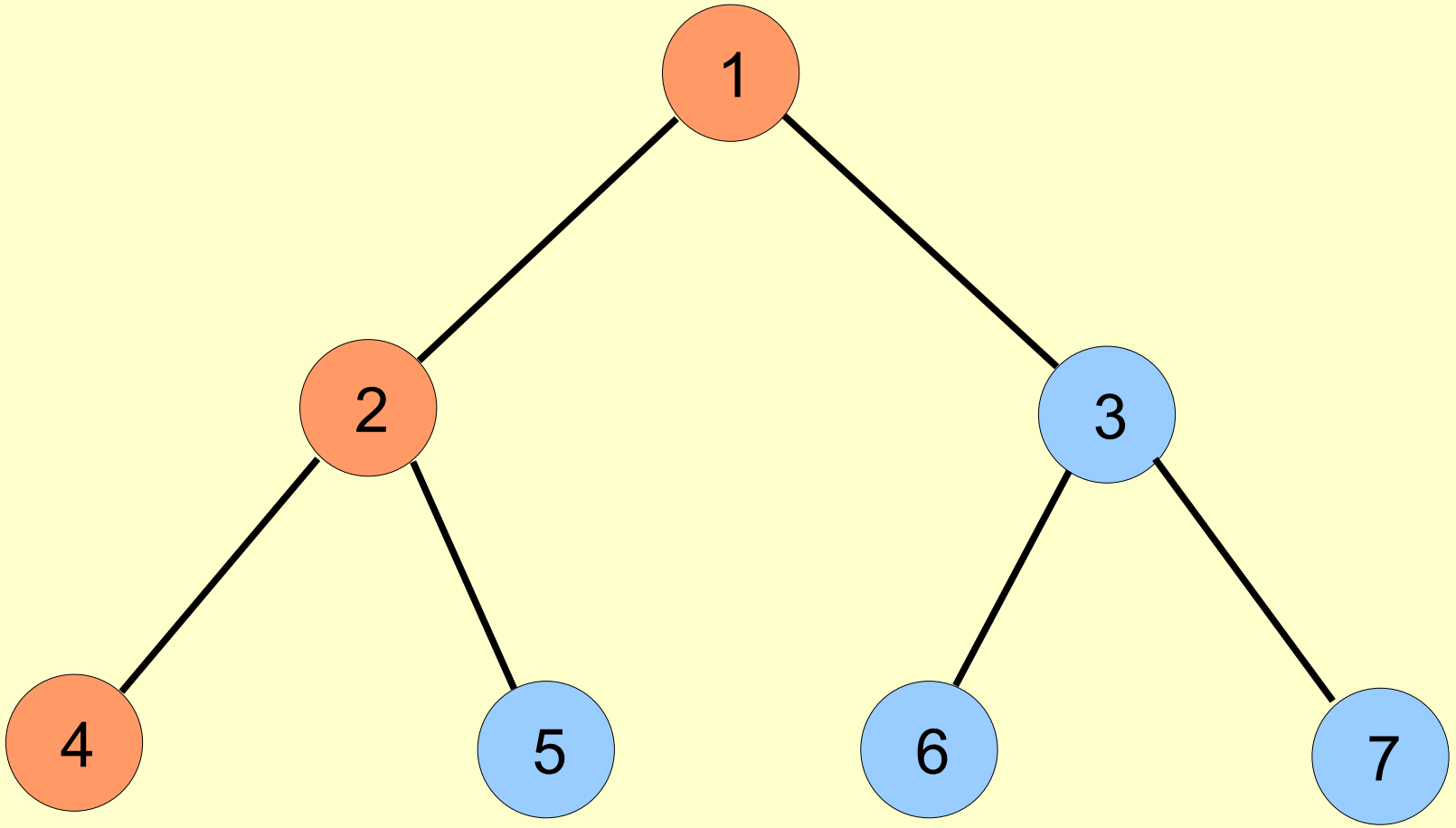
Quora

Tree

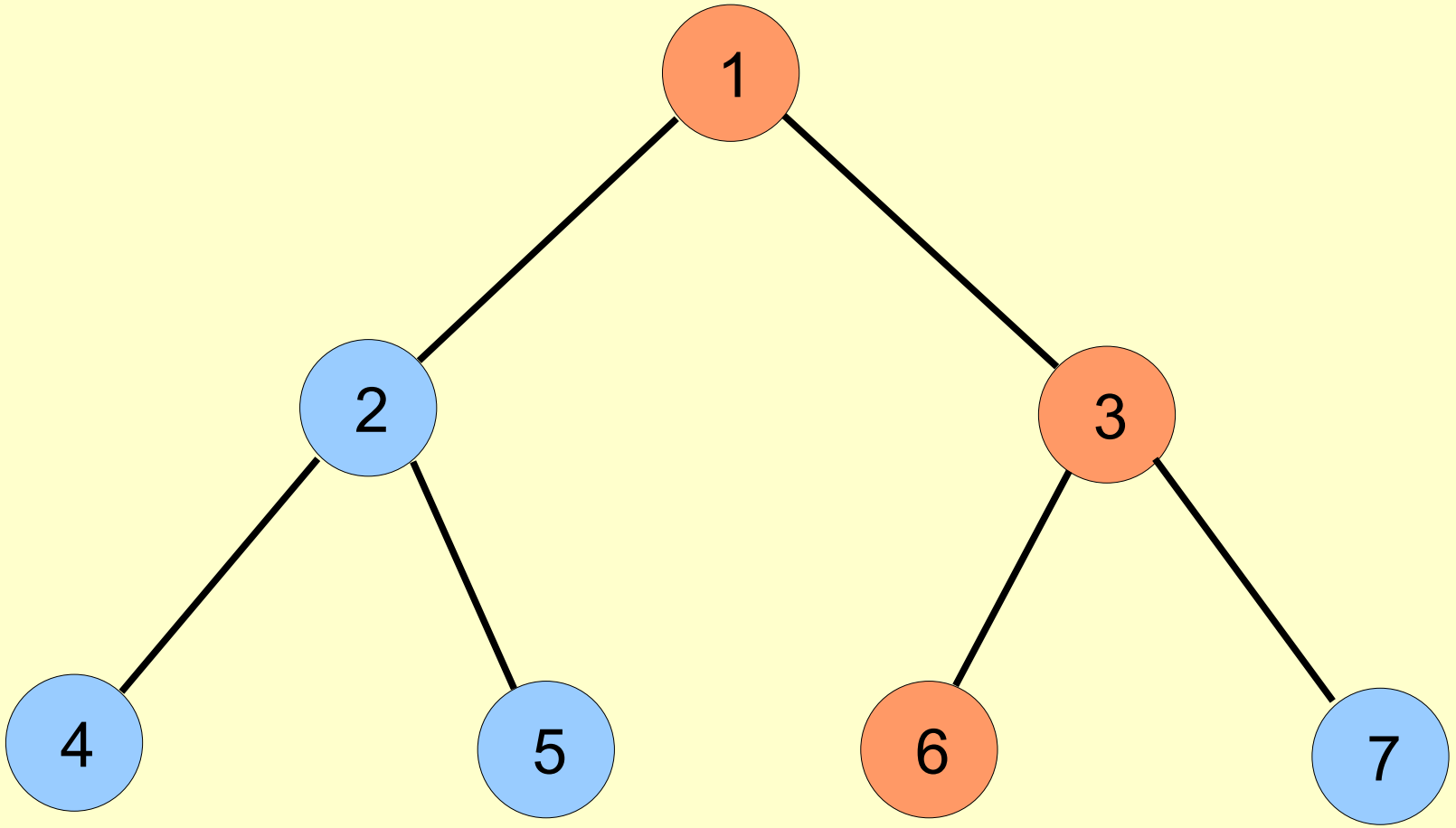


Quora

Tree

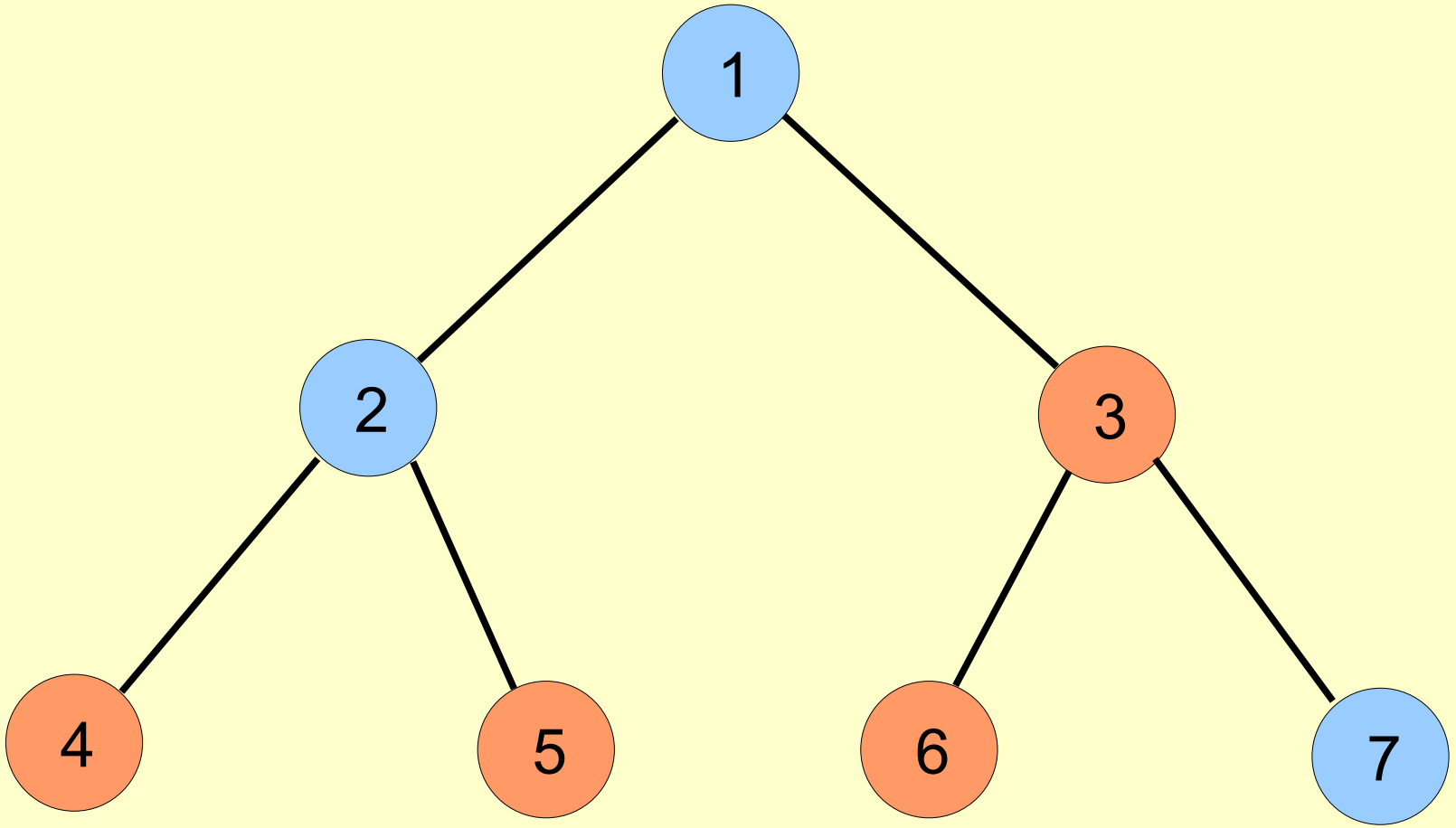


Quora



Quora

Tree



Quora

Agarwal, El Abbadi

```
function GetQorum (Tree:tree) : quorumset;
var left, right : quorumset;
begin
   if Empty(Tree) then
      return ({});
   else
      if GrantsPermission(Tree^.Node) then
            return ({Tree^.Node} U
GetQuorum(Tree^.LeftChild))
         or
            return ({Tree^.Node} U
GetQuorum(Tree^.RightChild))
      else
               (* … handling Tree^.Node failure … *)
end.
```