

# Parallel Random Access Machine

F.CS306 ПАРАЛЛЕЛ ПРОГРАММЧЛАЛ – Лекц 3

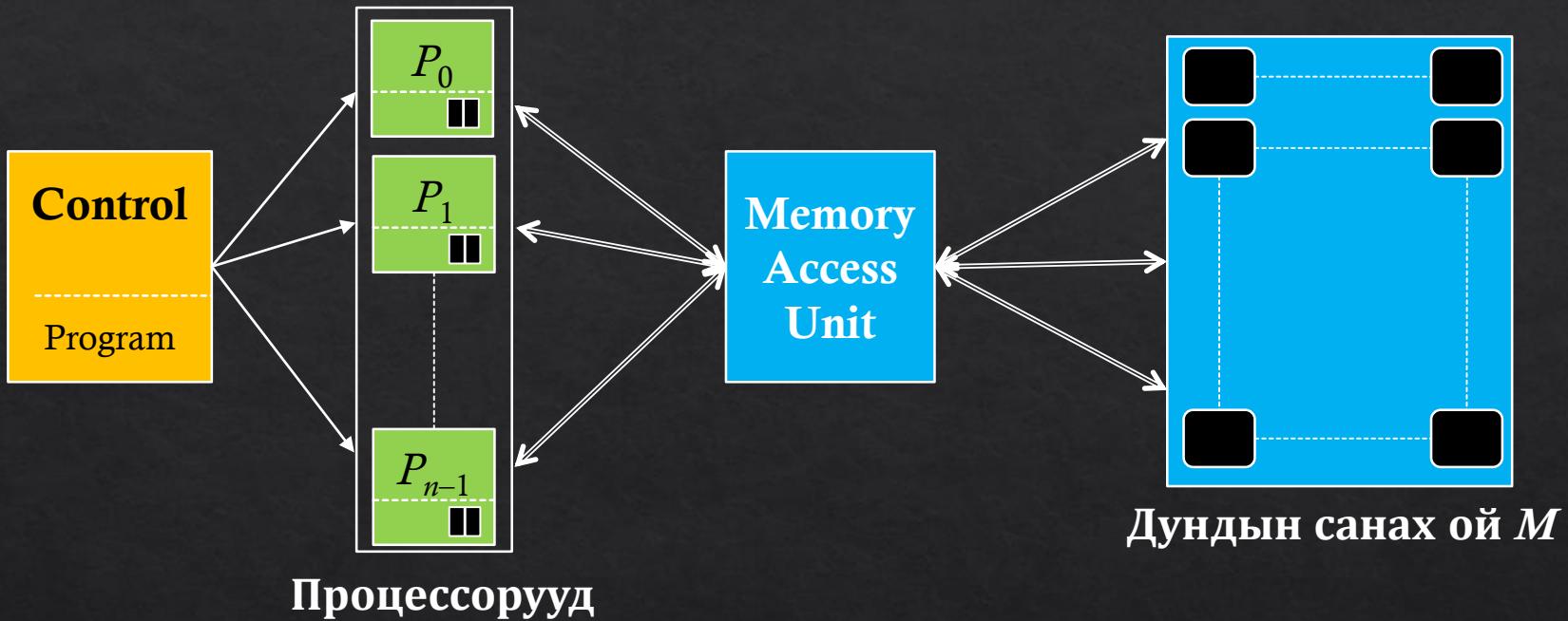
Г.ГАНБАТ

[ganbatg@must.edu.mn](mailto:ganbatg@must.edu.mn)

# Хичээлийн агуулга

- ❖ PRAM нь Дундын санах ойн тооцооллын загвар болох нь
- ❖ Түгээмэл PRAM алгоритмуудыг өртгийн хувьд шинжлэх, зарим cost-optimal PRAM алгортын зохиомжийг судлах
- ❖ Хуваарилагдсан санах ойн системийн болон сүлжээний архитектурын зарим энгийн топологуудыг мэдэх
- ❖ Эдгээр чанаруудыг графын онолын degree, bisection width, болон diameter дээр үндэслэн харьцуулах

# Parallel Random Access Machine (PRAM)

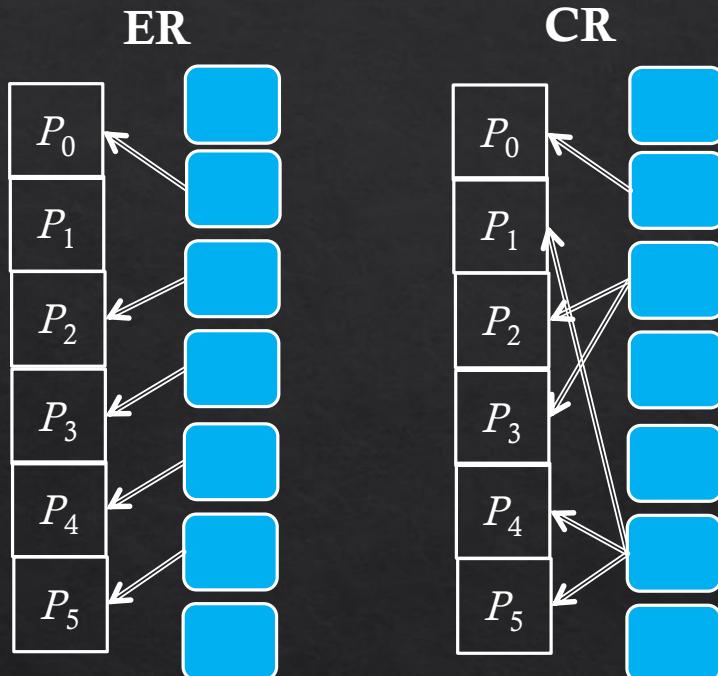


- ❖  $n$  ширхэг  $P_0, \dots, P_{n-1}$  процессорууд  $M$  глобал дундын санах ойд холсогдсон
- ❖ Санах ойн байрлал бүр процессоруудаас ижилхэн тогтмол хандалттай
- ❖ Процессорууд хоорондоо дундын санах ойд уншиж, бичих байдлаар харилцана

# PRAM

- ❖  $n$  ширхэг  $P_i$ ,  $i = 0, \dots, n-1$  ижил процессорууд lock-step горимд ажилланана
- ❖ Процессорын алхам бүрт 3 шаттай зааврын цикл гүйцэтгэнэ:
  1. **Үншилт:** Процессор бүр нэг өгөгдлийг дундын санах ойн (ялгаатай) үүрнээс зэрэг унших боломжтой бөгөөд локал регистрт хадгална.
  2. **Тооцоолол:** Процессор бүр энэ локал өгөгдөл дээр үндсэн үйлдлийг гүйцэтгэж дараа нь үр дүнг буцааж регистрт хадгална.
  3. **Бичилт:** Процессор бүр өгөгдлийг дундын санах ойн үүрэнд зэрэг бичих боломжтой. PRAM хувилбар давхардсан бичилтийг зөвшөөрдөггүй бол зөвхөн өөр үүрэнд, зөвшөөрдөг бол ижил байрлалд бичилт хийнэ. (race conditions – уралдааны нөхцөл)

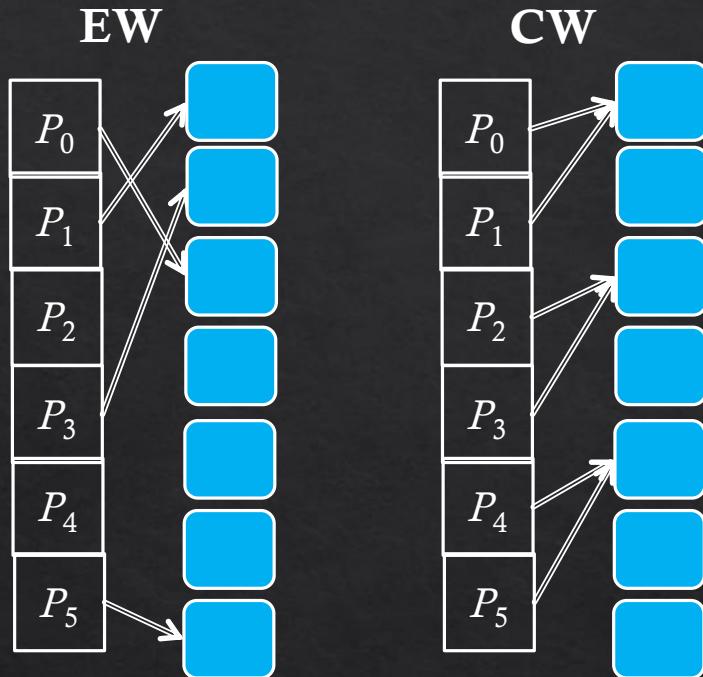
# PRAM хувилбарууд



- ❖ **Exclusive Read Exclusive Write (EREW):** Ямар ч циклийн явцад дундын санах ойн нэг үүрэнд хоёр процессор уншиж, бичихийг зөвшөөрдөггүй.
- ❖ **Concurrent Read Exclusive Write (CREW):** Зарим процессорууд дундын санах ойн нэг үүрнээс зэрэг унших боломжтой. Гэсэн ч өөр процессорууд дундын санах ойн нэг үүрэнд бичихийг зөвшөөрдөггүй.

CRCW PRAM зааврын циклийг  $O(1)$  тогтмол хугацаанд гүйцэтгэдэг

# PRAM хувилбарууд



- ◆ **Concurrent Read Concurrent Write (CRCW):** Дундын санах ойн үүрэнд зэрэгцээгээр уншиж бичдэг. Зэрэг бичилт хийх тохиолдолд (уралдааны нөхцөлд хүргэх) аль утыг нь хадгалахыг тодорхой зааж өгнө:
- 1. **Priority CW:** Процессорууд нь ялгаатай зэрэглэлтэй тодорхойлогдсон байх ба энд өндөр зэрэглэлтэй нь бичнэ.
- 2. **Arbitrary CW:** Бичих утгыг санамсаргүйгээр сонгоно.
- 3. **Common CW:** Бүх утгууд тэнцүү бол энэ утгыг бичнэ, үгүй санах ойн байрлалд өөрчлөлт орохгүй.
- 4. **Combining CW:** Бүх утггуудыг хоёртын үйлдлээр нэгтгээд бичнэ. (Жнь. нийлбэр, үржвэр, хамгийн бага, логик AND)
- ◆ EREW, CREW PRAM зааврын циклийг  $\lceil \log_2(n) \rceil + 1$  гүйцэтгэдэг

# Паралел Prefix тооцоолол

- ❖  $\mathcal{X}$  олонлог дээрх  $\circ$  хоёртын үйлдэл. Жны:  $\circ: \mathcal{X} \times \mathcal{X} \rightarrow \mathcal{X}$ 
  - ❖  $x_i, x_j, x_k \in \mathcal{X}$  байх  $(x_i \circ x_j) \circ x_k = x_i \circ (x_j \circ x_k)$ .
  - ❖ Жны: Нэмэх, Үржих, Хамгийн бага, Тэмдэгт мөр залгах, Булийн AND/OR
- ❖  $i = 0, \dots, n-1$  байх  $X = \{x_0, \dots, x_{n-1}\}$ ,  $x_i \in \mathcal{X}$ . Дараах байдлаар тооцоолно
  - ❖  $s_0 = x_0$
  - ❖  $s_1 = x_0 \circ x_1$
  - ❖ ...
  - ❖  $s_n = x_0 \circ x_1 \circ \dots \circ x_{n-1}$ .
- ❖  $X = \{x_0, \dots, x_{n-1}\}$  -ээс  $S = \{s_0, \dots, s_{n-1}\}$  олох гэдэг нь **prefix тооцоолол**
- ❖ Жишээ нь: “Хамгийн бага”
  - ❖ Оролт:  $\{39, 21, 20, 50, 13, 18, 2, 33, 49, 39, 47, 15, 30, 47, 24, 1\}$
  - ❖ Гаралт:  $\{39, 21, 20, 20, 13, 13, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1\}$
- ❖ Тооцооллын хүндрэлийн доод хязгаар:  $\Omega(n)$

# PRAM дээрх параллел Prefix нийлбэр

- Энгийн давталтаар нийлбэр тооцоолох дараалсан алгоритм:

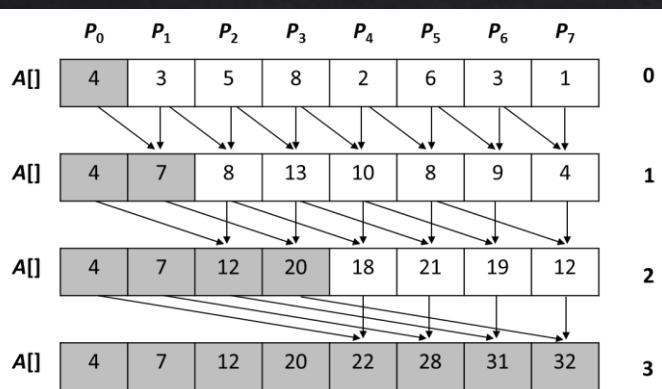
```
for (i=1; i<n; i++) A[i] = A[i] + A[i-1];
```

$$C(n) = T(n,p) \times p = O(n)$$

$$C(n) = T(n,1) \times 1 = O(n) \times 1 = O(n)$$

- “recursive doubling”-д суурилсан давталттай параллел Prefix нийлбэр

```
for (j = 0; j<n; j++) do_in_parallel // j процессор бүр
    reg_j = A[j];
for (i = 0; i<ceil(log(n)); i++) do // гадаад дараалсан давталт
    for (j = pow(2, i); j<n; j++) do_in_parallel // j процессор бүр
        reg_j += A[j - pow(2, i)]; // тооцоолол гүйцэтгэх
        A[j] = reg_j; // үр дүнг дундын санах оид хадгалах
}
```



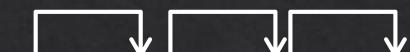
- Псевдо код:  $\lceil \log_2 n \rceil$  давталт гүйцэтгэнэ
- $C(n) = T(n,p) \times p = O(\log(n)) \times n = O(n \times \log(n))$
- Cost-optimal “биш”

# Cost-Optimal PRAM Parallel Prefix

- log-linear-aac linear болгох
  - Ажиллах хугацааг багасгах (хүнд)
  - $p = n$  процессорын тоог бууруулах:  $p = \frac{n}{\log_2(n)}$
- Алгоритмын бүдүүвч
  1. Оролтын  $n$  ширхэг утгыг  $\log_2(n)$  хэмжээтэй хэсгүүдэд хуваана. Процессор бүр нэг хэсэг дээр Prefix нийлбэрийг параллелаар тооцоолно:  $O(\log(n))$ .
  2.  $\frac{n}{\log_2(n)}$  хэсэг бүр дээр өмнөх Prefix нийлбэрийн алгоритмыг гүйцэтгэнэ:  $O(\log(\frac{n}{\log(n)}))$
  3. Процессор бүр 2-р шатанд тооцоолсон зүүн талын утгыг өөрийн хэсгийн бүх утгууд дээр нэмнэ:  $O(\log(n))$ .
- *Local Register* шаардлагагүй
- $C(n) = T(n,p) \times p = O(\log(n)) \times O(\log(\frac{n}{\log(n)})) = O(n)$

# Cost-Optimal PRAM Parallel Prefix

P0	0	1	2	3
P1	4	5	6	7
P2	8	9	10	11
P3	12	13	14	15



P0	0	1	3	6
P1	4	9	15	22
P2	8	17	27	38
P3	12	25	39	54

P0	0	1	3	6
P1	4	9	15	28
P2	8	17	27	66
P3	12	25	39	120



P0	0	1	3	6	
P1	10	15	21	28	
P2	36	45	55	66	
P3	78	91	105	120	

# EREW PRAM дээрх Cost-optimal параллел Prefix

```
// 1 шат: i процессор бүр n/p = log(n) = k хэмжээтэй дэд массивийн локал
// prefix нийлбэрийг тооцоолно
for (i = 0; i<n / k; i++) do_in_parallel
    for (j = 1; j<k; j++) do
        A[i*k+j] += A[i*k+j-1];

// 2 шат: дэд массив бүрийн баруун талын эхний утгыг хэрэглэн Prefix
// нийлбэрийн олно (O(log(n/k)))
for (i = 0; i<log(n / k); i++) do
    for (j = pow(2, i); j<n / k; j++) do_in_parallel
        A[j*k-1] += A[(j-pow(2, i))*k-1];

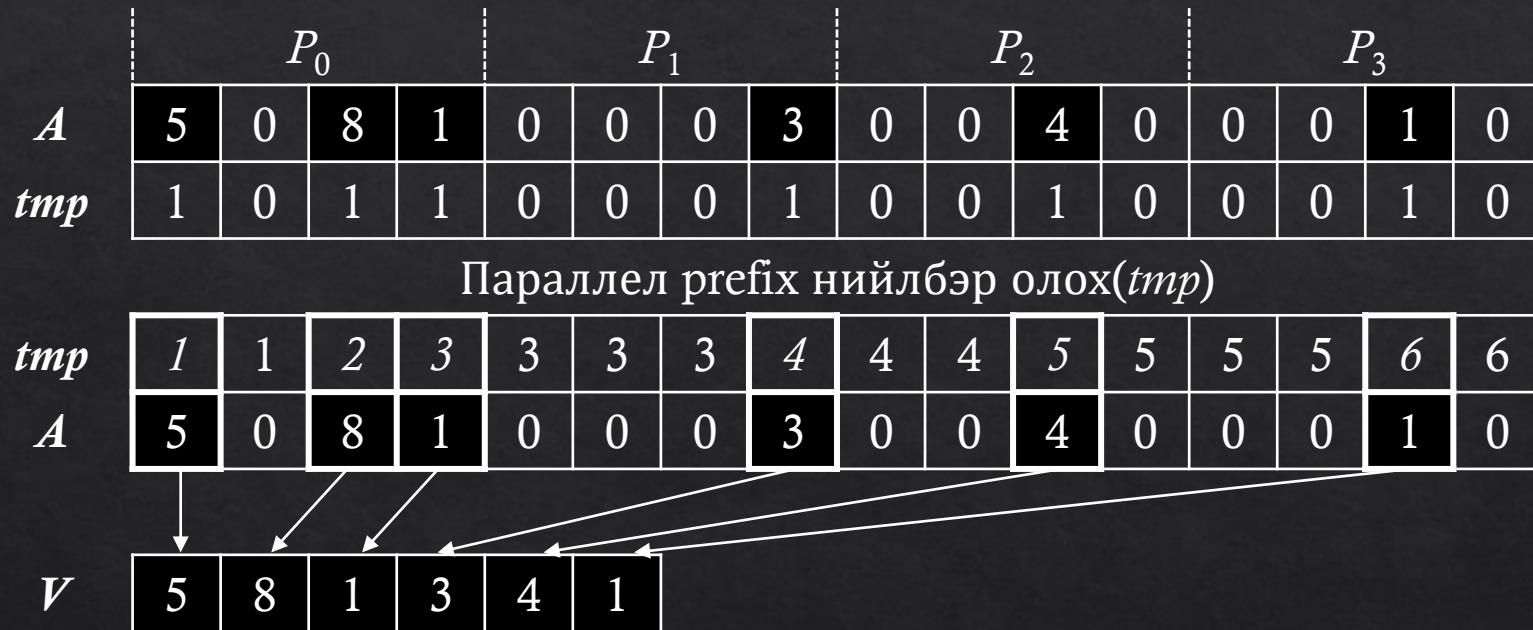
// 3 шат: i процессор бүр 2 шат дээр тооцоолсон i-1 дугаартай утгыг дэд
// массивийн сүүлийнхээс бусад элемент дээр нэмнэ.
for (i = 1; i<n / k; i++) do_in_parallel
    for (j = 0; j<k - 1; j++) do
        A[i*k+j] += A[i*k+j-1];
```

- $n = 2^k$  байвал  $p = \frac{n}{k}$

# PRAM Сийрэг массив нягтруулах

- ❖ Ихэнх гишүүн нь 0 байх 1 хэмжээст массив.
  - ❖ 0-ээс бусад утгууд ( $V$  массив), координатуудыг нь ( $C$  массив) хадгалж санах ойн хэмнэлтийг үүсгэж болно.
  - ❖ Дараалсан алгоритмаар  $A$ -ийн  $n$  элементийг зүүнээс баруун тийш давтаж,  $V$  ба  $C$ -г шугаман хугацаанд байгуулна.
  - ❖ Параллел Prefix аргаар  $p = \frac{n}{\log_2(n)}$  процессор ашиглан оновчтой PRAM алгоритмыг бичих боломжтой.
1.  $A[i] \neq 0$  бол  $\text{tmp}[i] = 1$ , үгүй бол  $\text{tmp}[i] = 0$  байх засврын tmp массив үүсгэнэ. Дараа нь tmp дээр Параллел Prefix нийлбэрийг гүйцэтгэнэ.  $A$ -ийн 0 биш элемент бурийн хувьд tmp-д хадгалагдсан утга нь  $V$  дахь тухайн элементийн хаяг байна.
  2. Параллел Prefix нийлбэрээр үүсгэгдсэн хаягуудыг ашиглан бид  $A$ -аас  $V$ -руу 0 биш элементүүдийг бичнэ. Харгалзах координатуудыг мөн байдлаар  $C$  руу бичнэ.

# PRAM Сийрэг массив нягтруулах



- ❖  $p = n/\log(n)$  ширхэг процессор
- ❖  $C(n) = T(n,p) \times p = O(\log n) \times n/\log(n) = O(n)$

# Linear Array



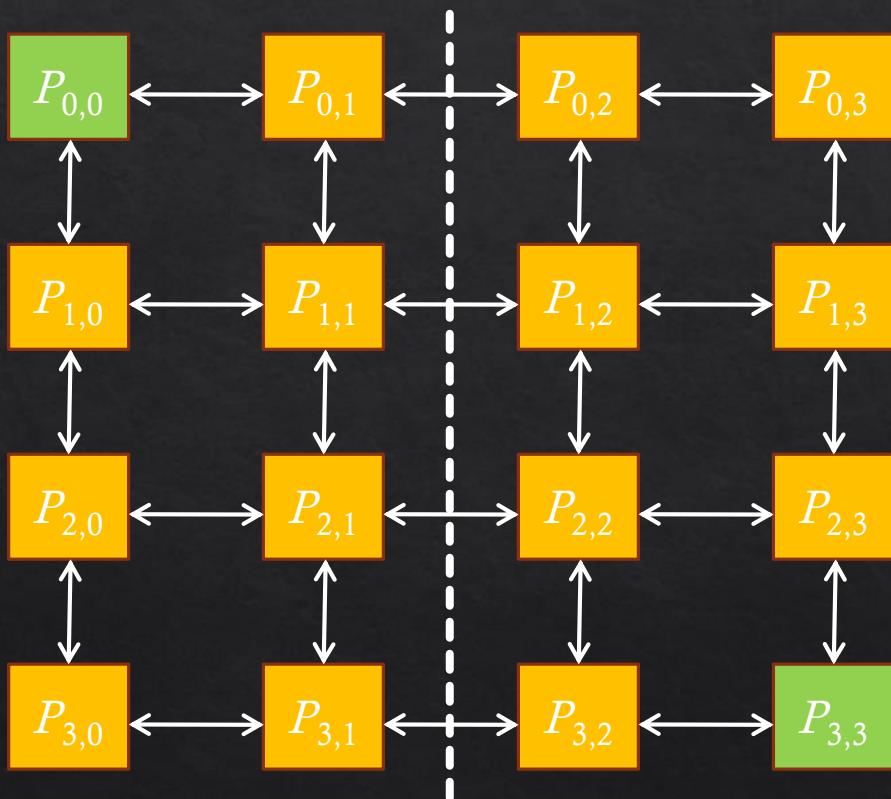
- ❖  $n$  процессортой шугаман массивийг  $L_n$  гэе.
- ❖ Сүлжээний **degree** нь процессоруудын хамгийн их хөршүүдийн тоо
  - ❖  $\text{degree}(L_n) = 2$
- ❖ Сүлжээний **diameter** нь холбоотой хоёр процессорын хоорондох хамгийн их зайд
  - ❖  $\text{diameter}(L_n) = n-1$
- ❖ Сүлжээний **bisection-width** нь сүлжээг хоёр тэнцүү хуваахад устгах холбоосуудын хамгийн бага байх тоо.
  - ❖  $bw(L_n) = 1$

# Contradictory requirements

Сүлжэний зохиомж нь **зөрчилдөөнтай шаардлагуудын** хооронд уялдааг хангах эрмэлзэлтэй.

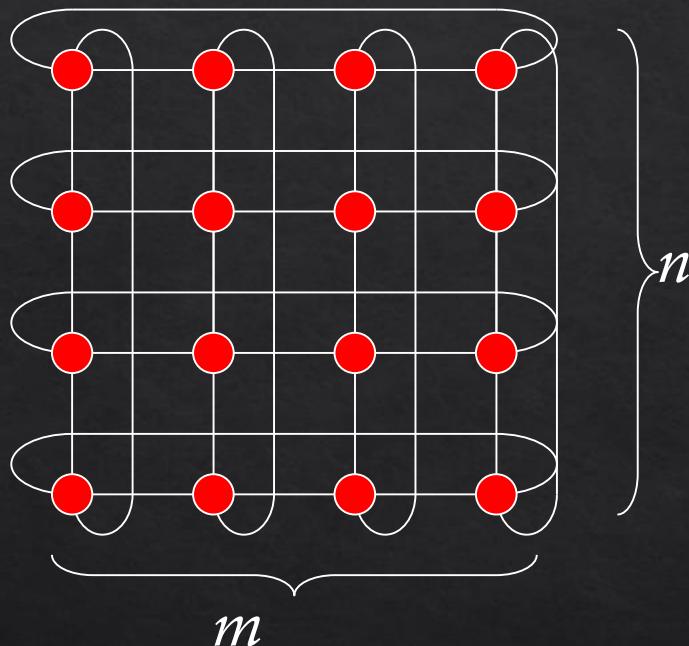
- ❖ **Constant degree:** Сүлжээний degree тогтмол байх ёсттой; Жны, Сүлжээг өргөтгөхөд нь холболтыг нэмэх шаадлагагүй.
- ❖ **Low diameter:** Аливаа хоёр процессорын холболтыг үр ашигтай байлгах үүднээс диаметрийг хамгийн бага байлгах хэрэгтэй.
- ❖ **High bisection-width:** bisection-width нь сүлжээний боломжит чадварыг илэрхийлэх бөгөөд энэ нь дотоод bandwidth-д нөлөөлнө.
  - ❖ Бага: харилцааг удаашруулснаар программын гүйцэтгэлийг хязгаарлана.
  - ❖ Их: Сүлжээний degree тогтмол бус байхыг шаардана.

# 2D Mesh



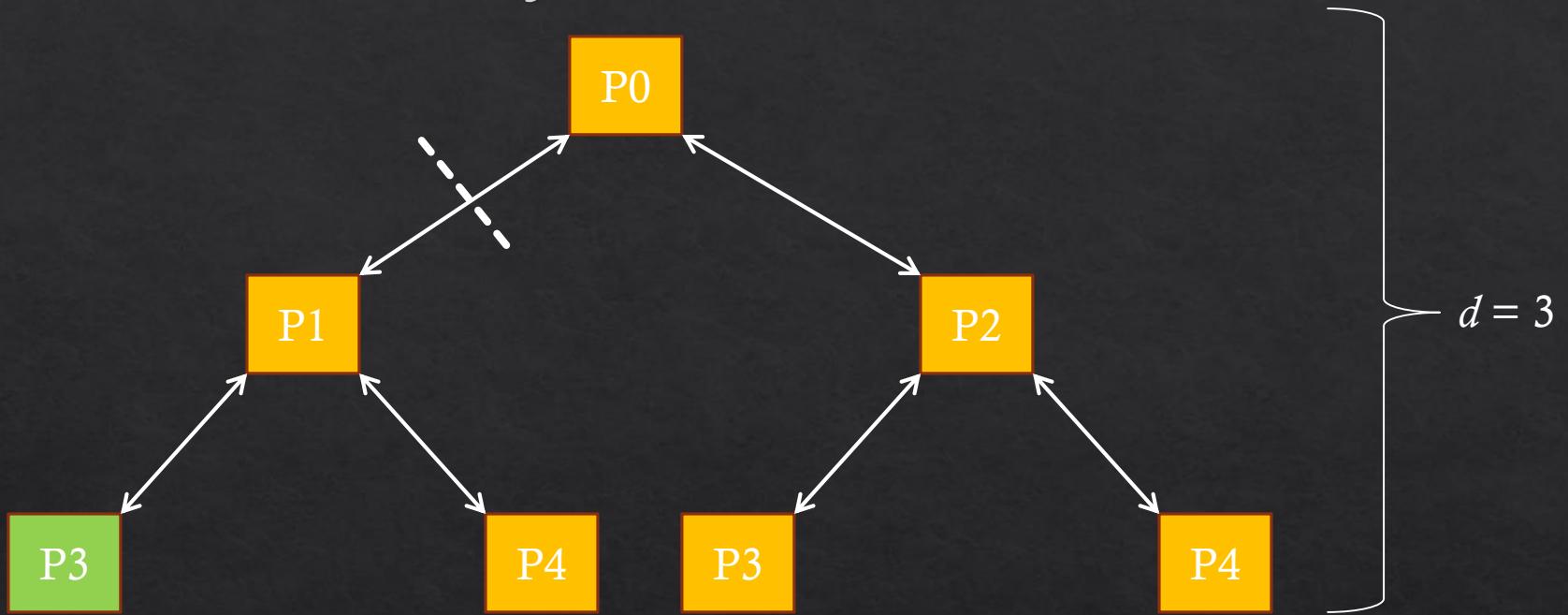
- ◆  $M(d,d)$  has  $n = d^2$  processors
- ◆  $\text{degree}(M(d,d)) = 4$ 
  - ◆  $O(1)$
- ◆  $\text{diameter}(M(d,d)) = 2(d-1)$ 
  - ◆  $O(\sqrt{n})$
- ◆  $\text{bw}(M(d,d)) = d$ 
  - ◆  $O(\sqrt{n})$

# 2D Torus



- ❖  $T(c,d)$  has  $n = c \cdot d$  processors
- ❖  $\text{degree}(T(c,d)) = 4$
- ❖  $\text{diameter}(T(c,d)) = d/2 + c/2$
- ❖  $\text{bw}(T(c,d)) = \min \{2 \cdot c, 2 \cdot d\}$

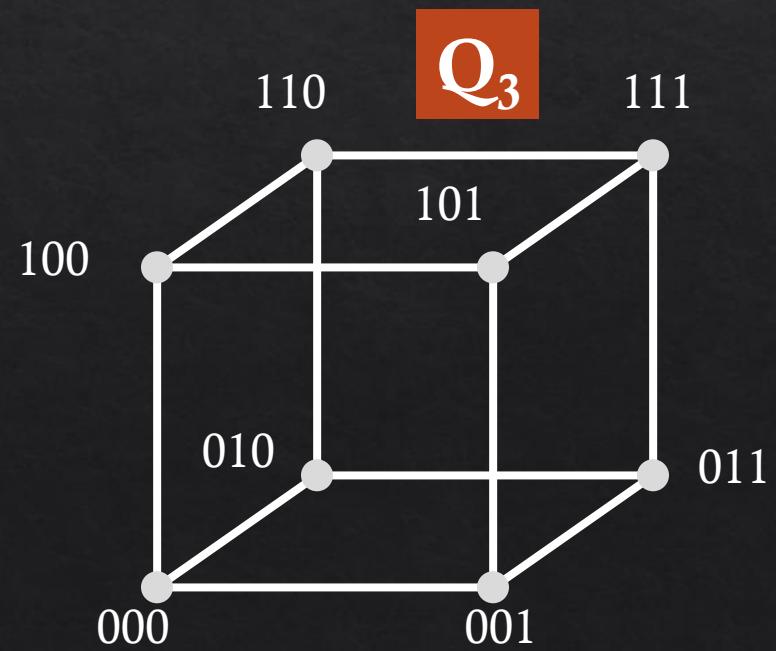
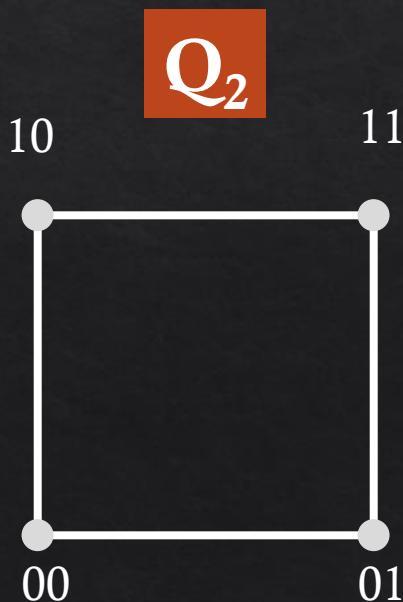
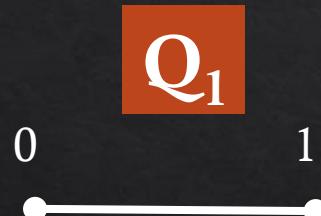
# Binary Tree



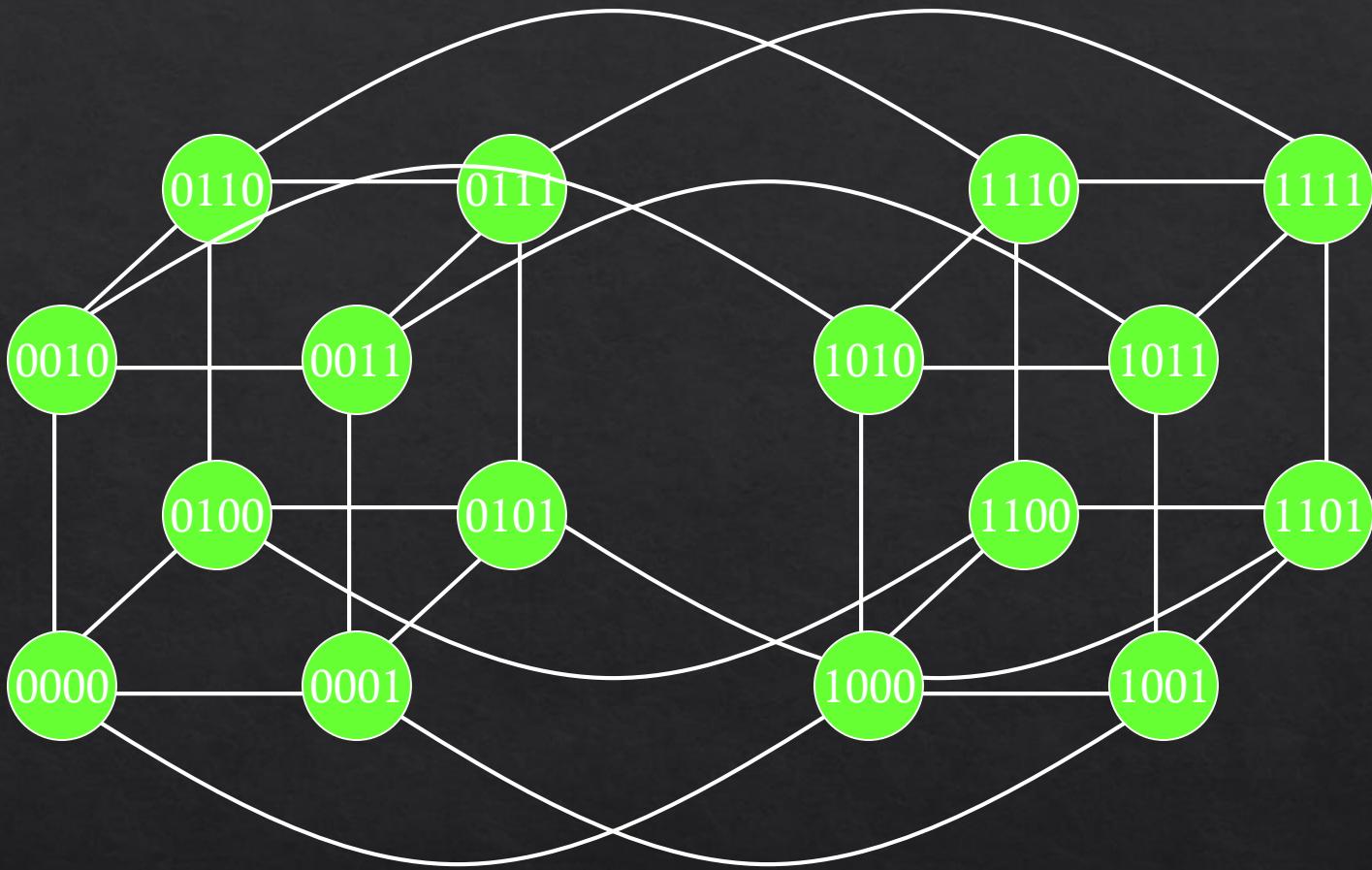
- ❖ BT( $d$ ) has  $n = 2^d - 1$  processors
- ❖ degree(BT( $d$ )) = 3
  - ❖ O(1)
- ❖ diameter(BT( $d$ )) =  $2(d-1)$ 
  - ❖ O(log( $n$ ))
- ❖ bw(BT( $d$ )) = 1
  - ❖ O(1)

# Hypercube

The **Hypercube**, denoted by  $Q_d$  ( $d \geq 1$ ), is the graph that has vertices representing the  $2^d$  bit strings of length  $d$ . Two vertices are adjacent if and only if the bit strings that they represent differ in exactly one bit position.



# Hypercube



- ❖  $\text{HC}(d)$  = Hypercube of degree  $d$ 
  - ❖ Number of processors:  $n = 2^d$
  - ❖  $\text{degree}(\text{HC}(d)) = d = O(\log(n))$
  - ❖  $\text{diameter}(\text{HC}(d)) = d = O(\log(n))$
  - ❖  $\text{bw}(\text{HC}(d)) = n/2 = O(n)$

# СҮЛЖЭЭНИЙ ТОПОЛОГИЙГ ҮНЭЛЭХ ШАЛГУУРУУД

Topology	Degree	Diameter	Bisection-Width
Linear Array	$O(1)$	$O(n)$	$O(1)$
2D Mesh/Torus	$O(1)$	$O(\sqrt{n})$	$O(\sqrt{n})$
3D Mesh/Torus	$O(1)$	$O(\sqrt[3]{n})$	$O(n^{2/3})$
Binary Tree	$O(1)$	$O(\log(n))$	$O(1)$
Hypercube	$O(\log(n))$	$O(\log(n))$	$O(n)$

## ❖ **Бага Diameter**

- ❖ Хос процессор бүрт үр ашигтай мэдээлэл солилцоог дэмжих зорилгоор

## ❖ **Их Bisection Width:**

- ❖ bisection width бага байх нь ачаалал ихтэй мэдээлэл солилцооны удаашруулж болох тул ингэснээр програмын чадавхийг хязгаарлаж болно
- ❖ Гэсэн ч, bisection width их байх нь non-constant degree шаардах боломжтой

## ❖ **Тогтмол** degree (i.e. independent of network size)

- ❖ Хэтэрхий олон холболт нэмэх шаардлагагүйгээр сүлжээг их хэмжээний зангилаатай болгон өргөтгөх боломжийг олгоно.

Баярлалаа.