Základní komunikační operace

Základní komunikační operace mezi procesy

- jde o komunikační vzory, které se v paralelním programování vyskytují velice často
- komunikující procesy mohou běžet na jednom nebo i na více výpočetních uzlech
- jejich speciální implemetace má následující výhody
  - zjednodušuje práci programátora
  - mohou využívat hardwarovou podporu dané paralelní architektury
  - udržují aktivní síťové spojení komunikační sítě
    - tím vlastně několik malých zpráv "shlukuje do jedné"
- k většině těchto operací existují i operace duální, které "běží v opačném směru"

Operace send a recieve

Jde o základní operace pro odeslání a přijetí jedné zprávy.

- `send( void *sendbuf, int nelems, int dest)`
  - `sendbuf` - ukazatel na pole dat, jež mají být odeslána
  - `nelems` - počet prvků pole
  - `dest` - ID procesu/uzlu, kterému mají být data zaslána
- `receive( void *recvbuf, int nelems, int source)`
  - `recvbuf` - ukazatel na pole, do nějž mají být přijatá data uložena
  - `nelems` - počet prvků, které budou přijaty
  - `source` - ID procesu/uzlu, od kterého mají být data načtena

Operace send a recieve

Příklad:

P_1	P_2
`a=100;`	
`send(&a,1,1);`	`receive( &a,1,0);`
`a=0;`	`printf("%d",a);`

Jaký výsledek vypíše proces P_2?

Operace send a recieve

Příklad:

P_1	P_2
`a=100;`	
`send(&a,1,1);`	`receive( &a,1,0);`
`a=0;`	`printf("%d",a);`

Jaký výsledek vypíše proces P_2?
Výsledek je `100` nebo `0`.

Operace send a recieve

- síťové rozhraní může využívat DMA a pracovat nezávisle na CPU
- pokud má P_2 zpoždění ve zpracování kódu, mohou být data z P_1 odeslána až po provedení příkazu `a=0;` na P_1
- tím pádem bude procesu P_2 odesláno číslo `0`

Blokující a neblokující posílání zpráv

Existují dva způsoby posílání zpráv:

- blokující
  - funkce `send` nevrátí řízení programu, dokud to není sémanticky bezpečné
- neblokující
  - komunikační funkce vrací řízení programu dříve, než je to sémanticky bezpečné

Blokující posílání zpráv

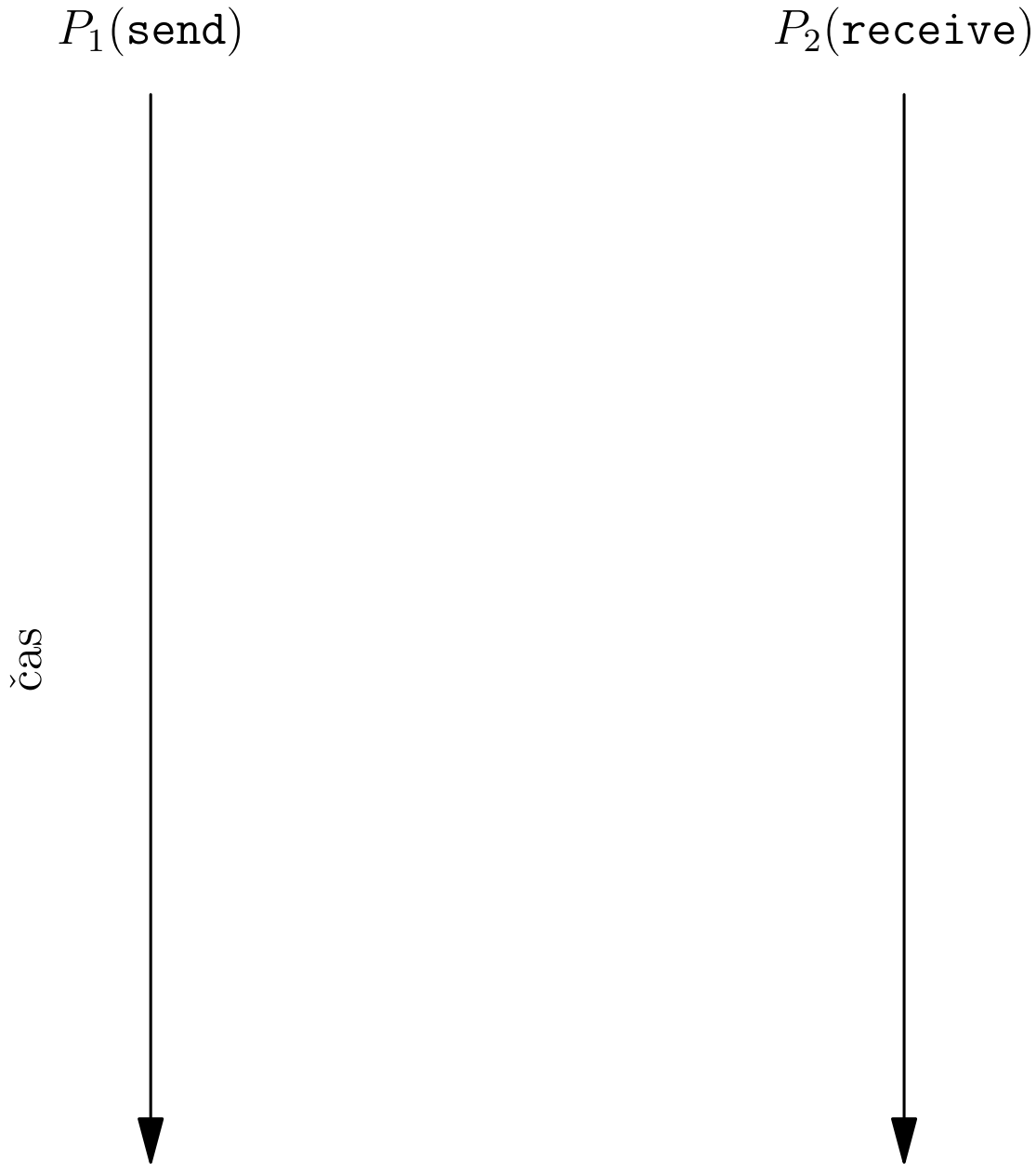
Blokující posílání zpráv je možné provést dvěma způsoby:

- bez bufferu
- s bufferem

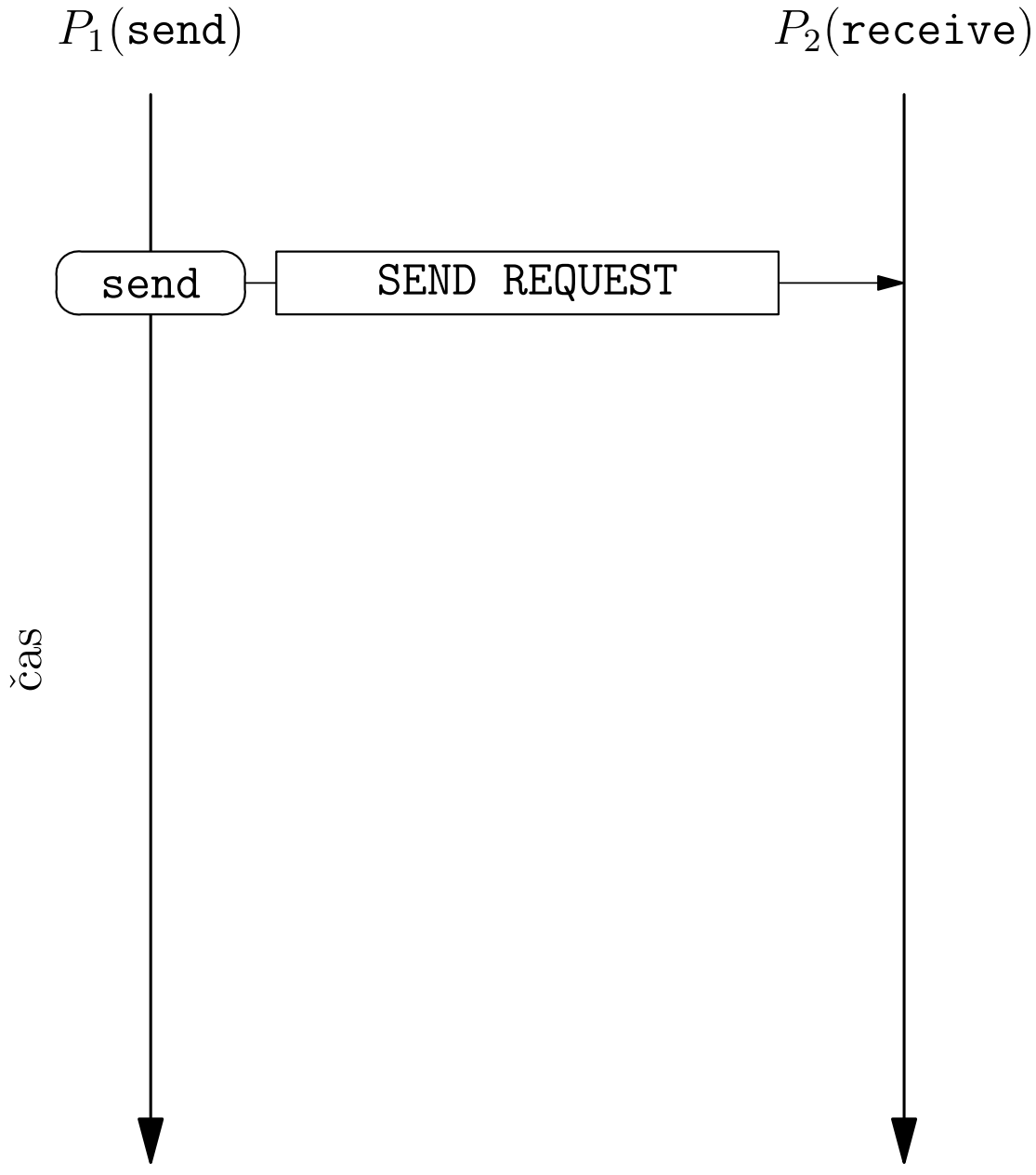
Blokující posílání zpráv bez bufferu

- odesílatel pošle požadavek k příjemci
- odesílatel potvrdí ve chvíli, kdy ve zpracování kódu dospěje k místu pro příjetí zprávy
- následně dojde k přenosu dat

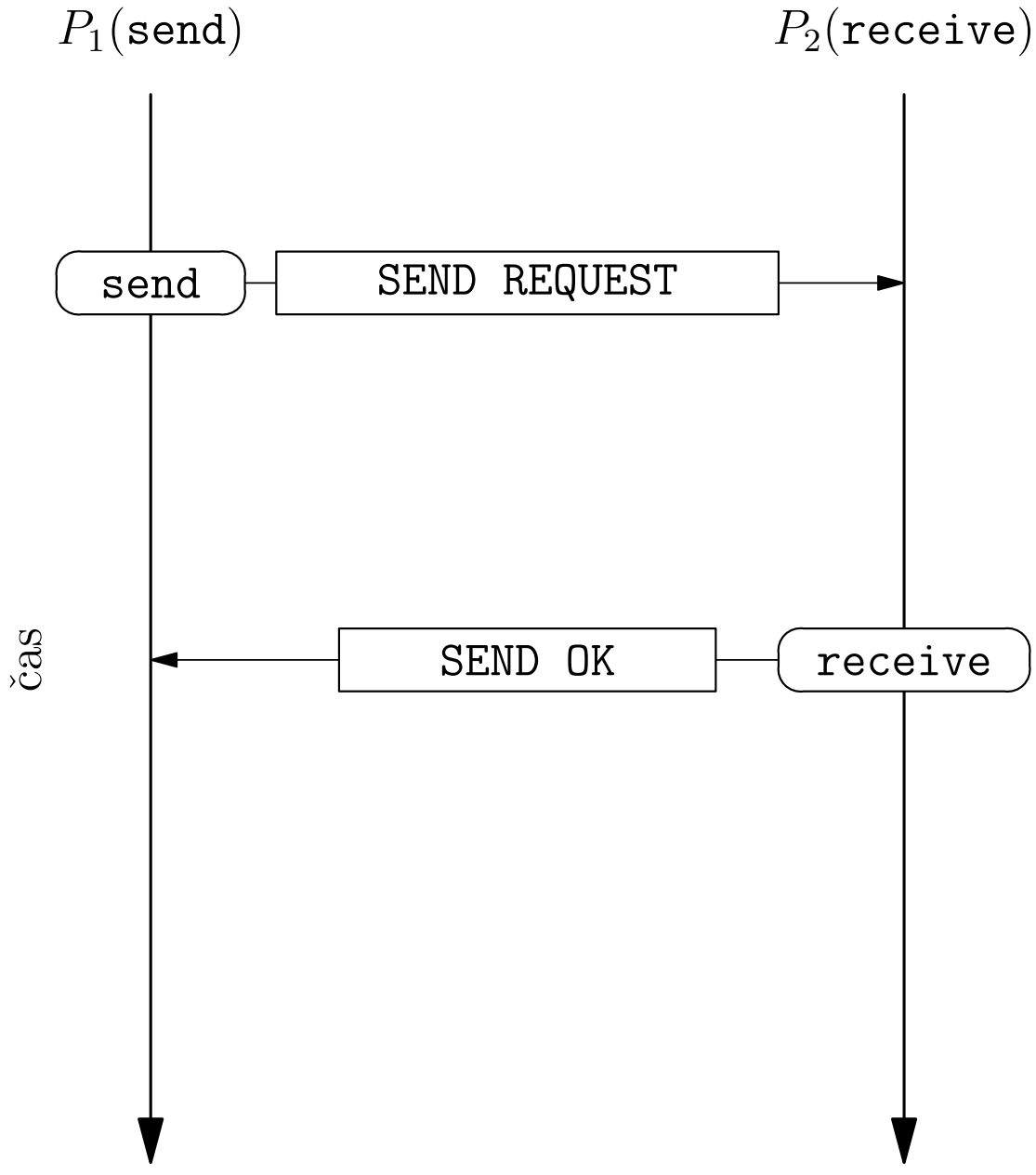
Blokující posílání zpráv bez bufferu



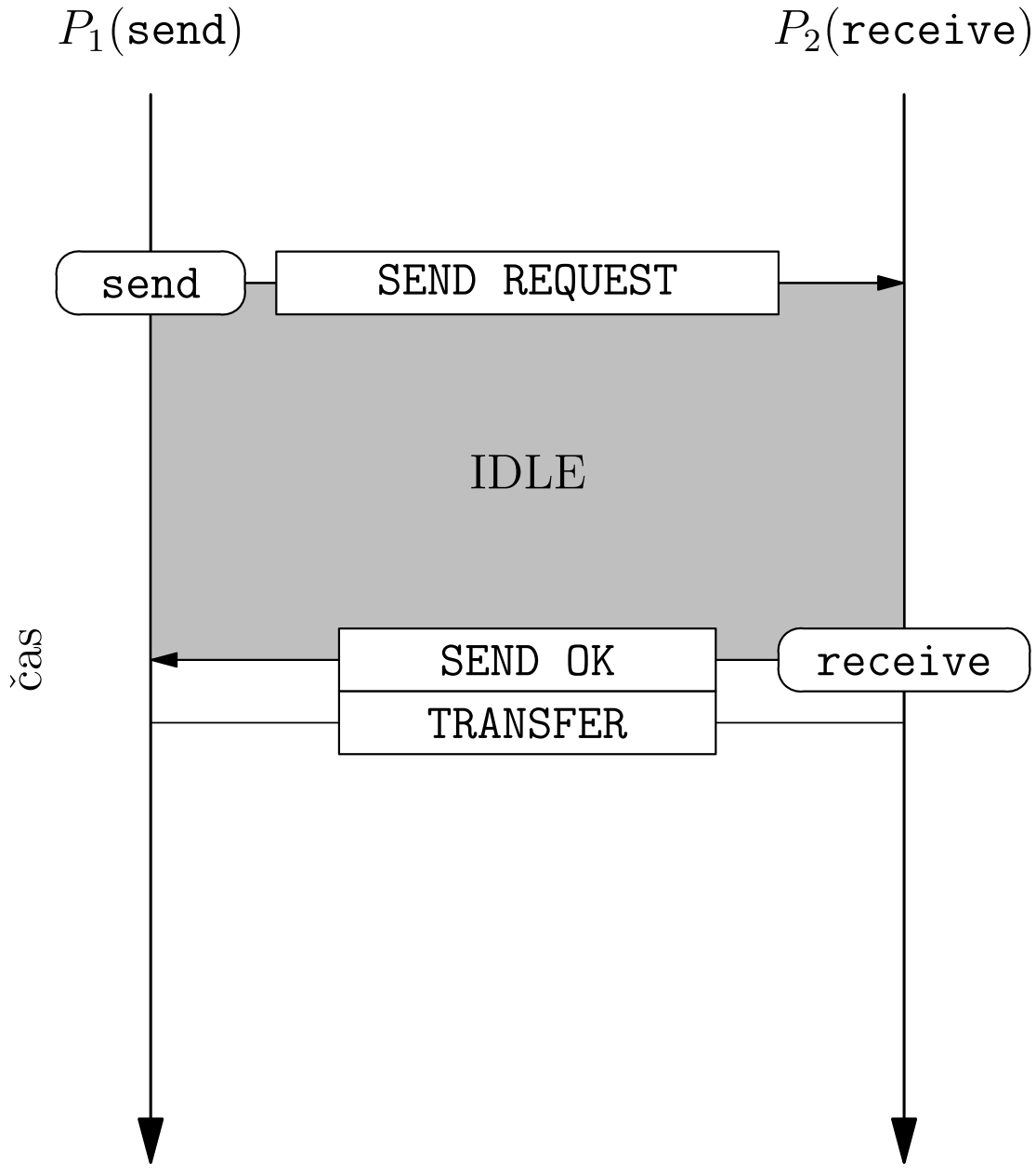
Blokující posílání zpráv bez bufferu



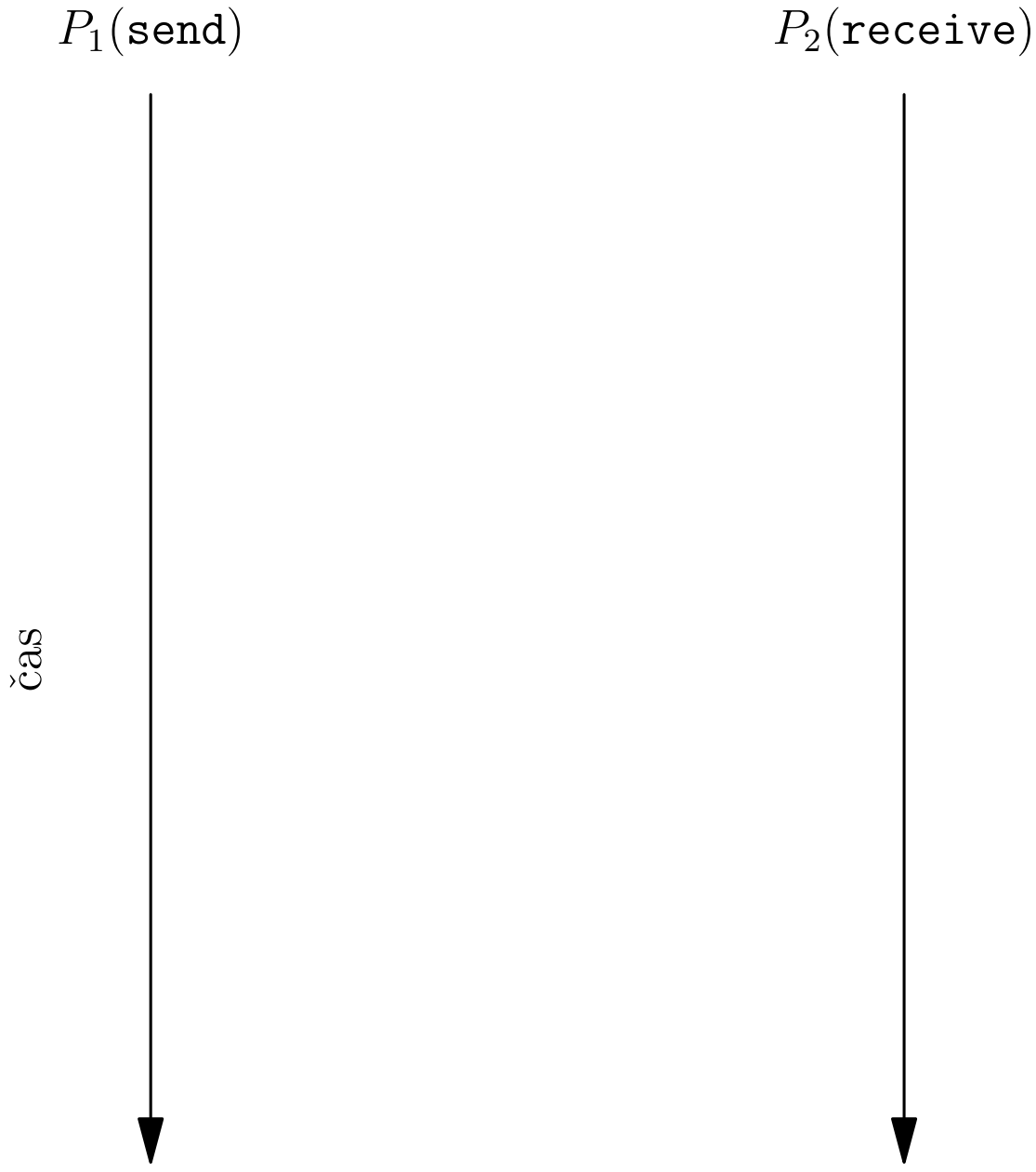
Blokující posílání zpráv bez bufferu



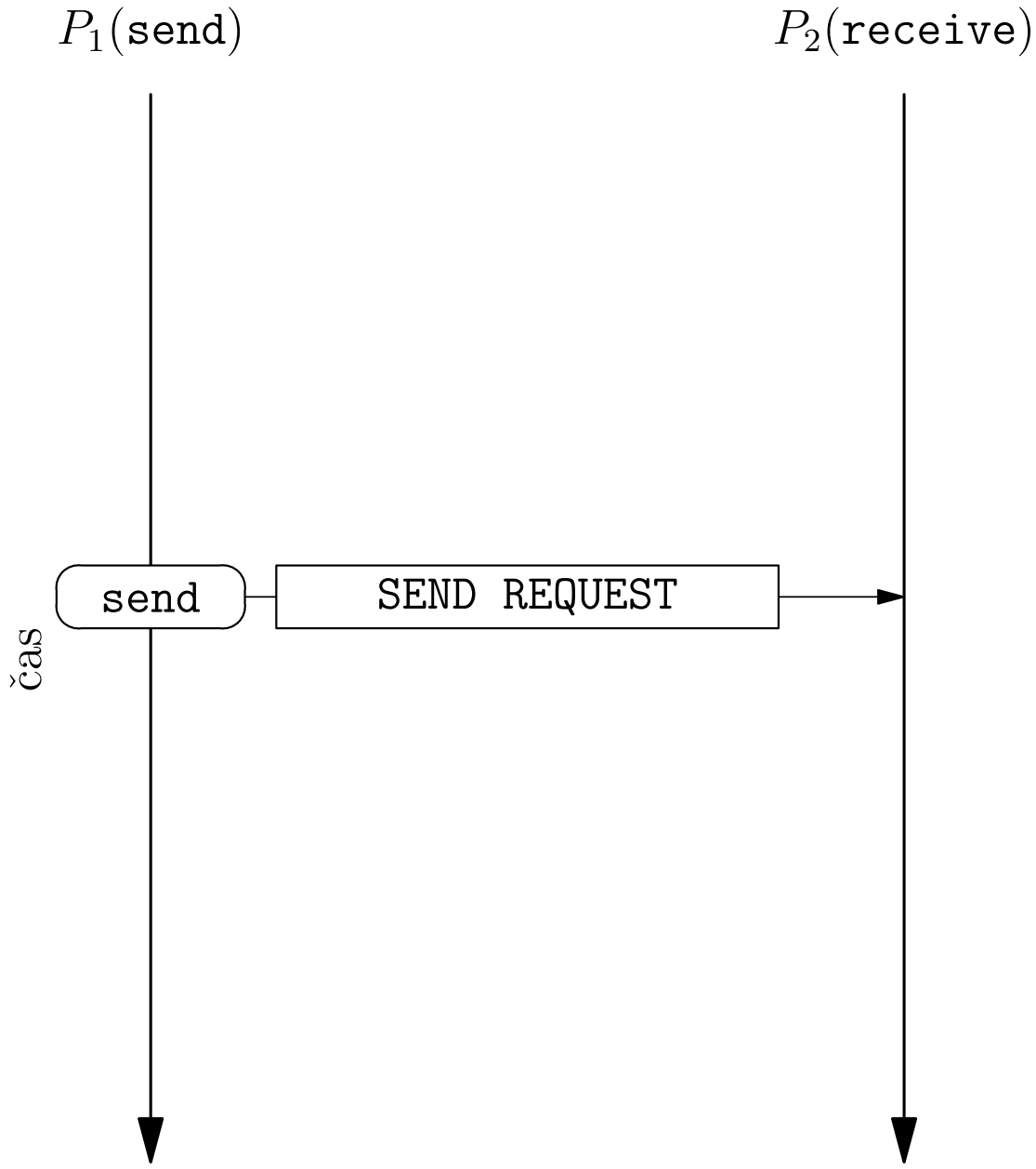
Blokující posílání zpráv bez bufferu



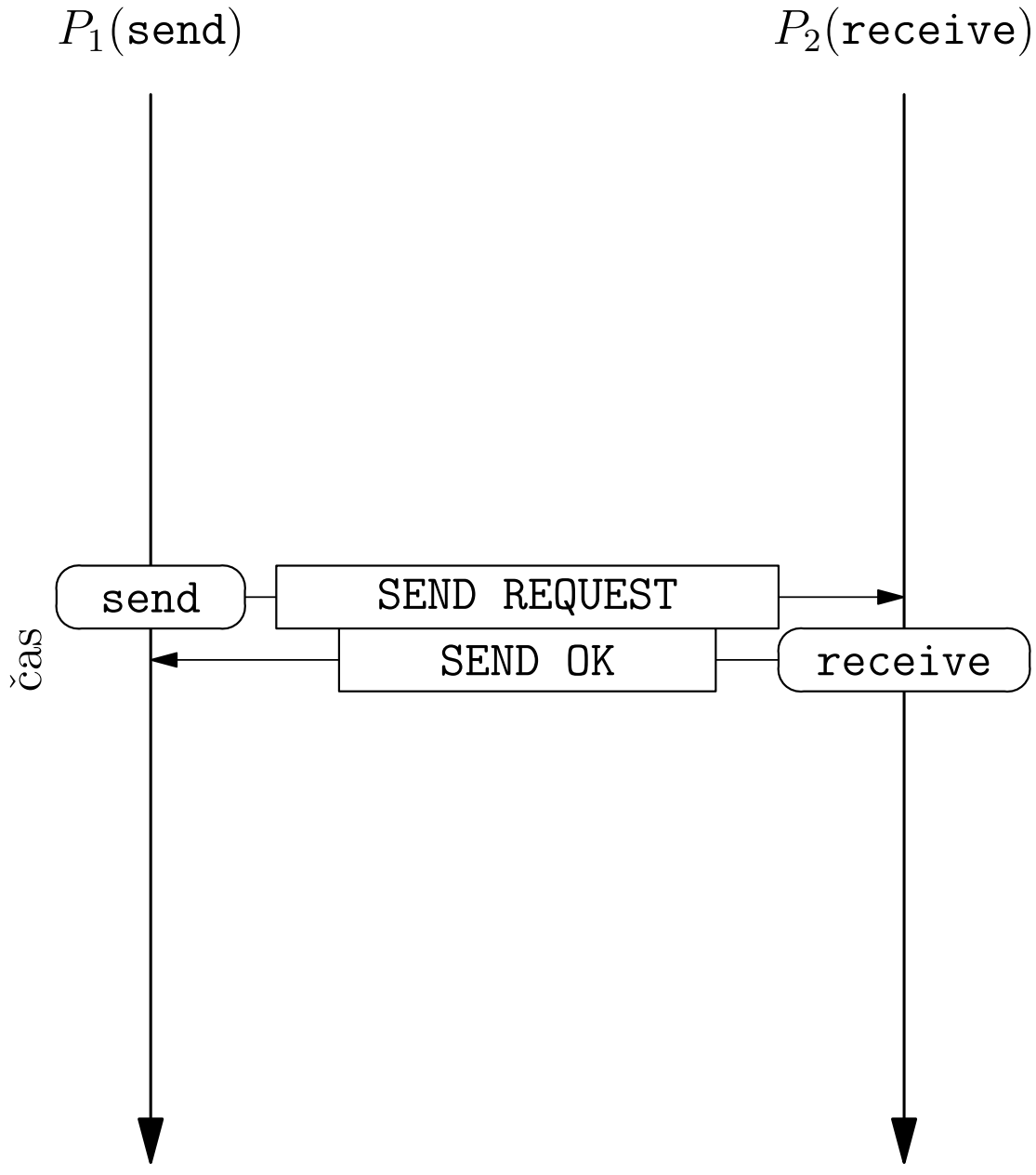
Blokující posílání zpráv bez bufferu



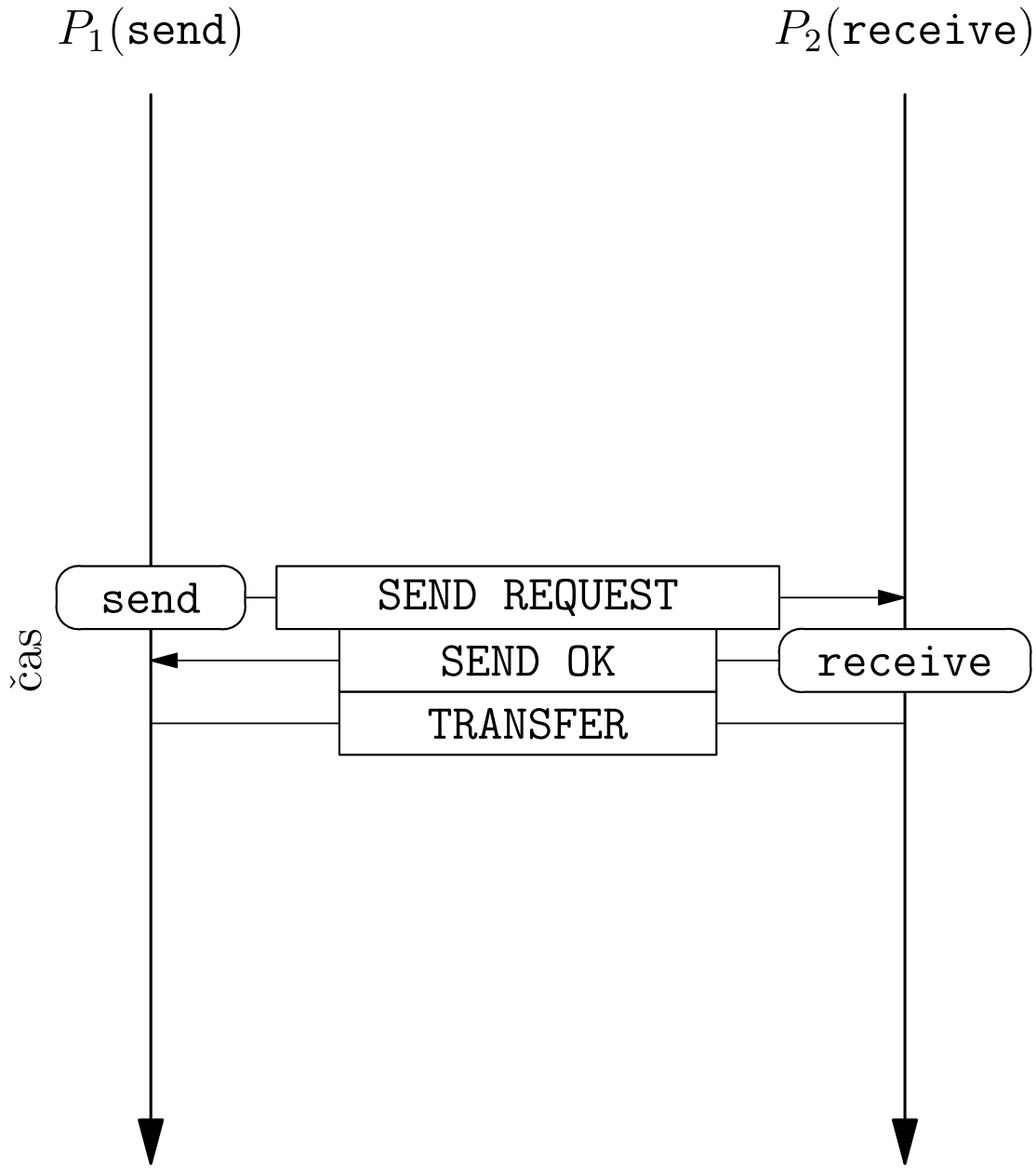
Blokující posílání zpráv bez bufferu



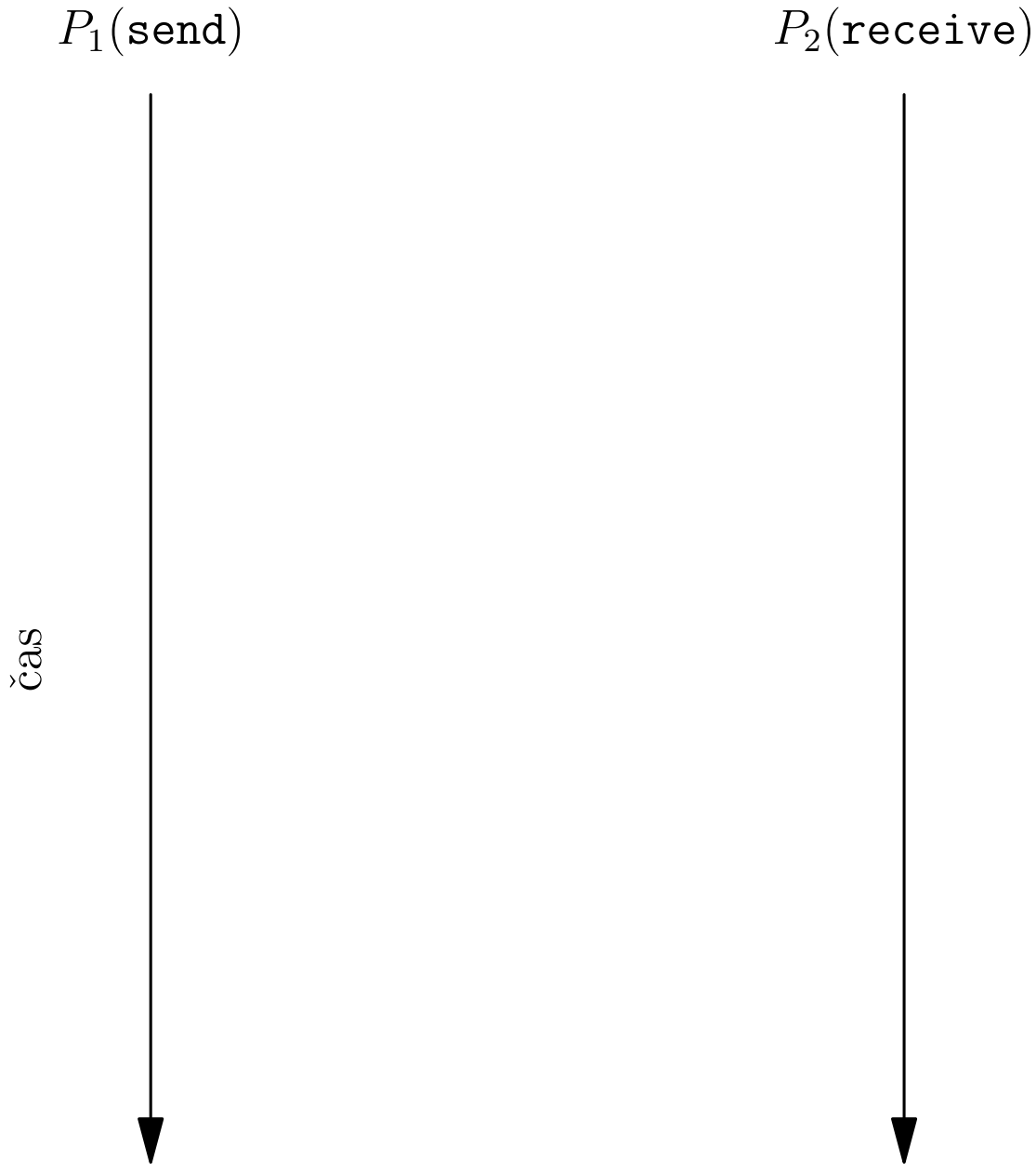
Blokující posílání zpráv bez bufferu



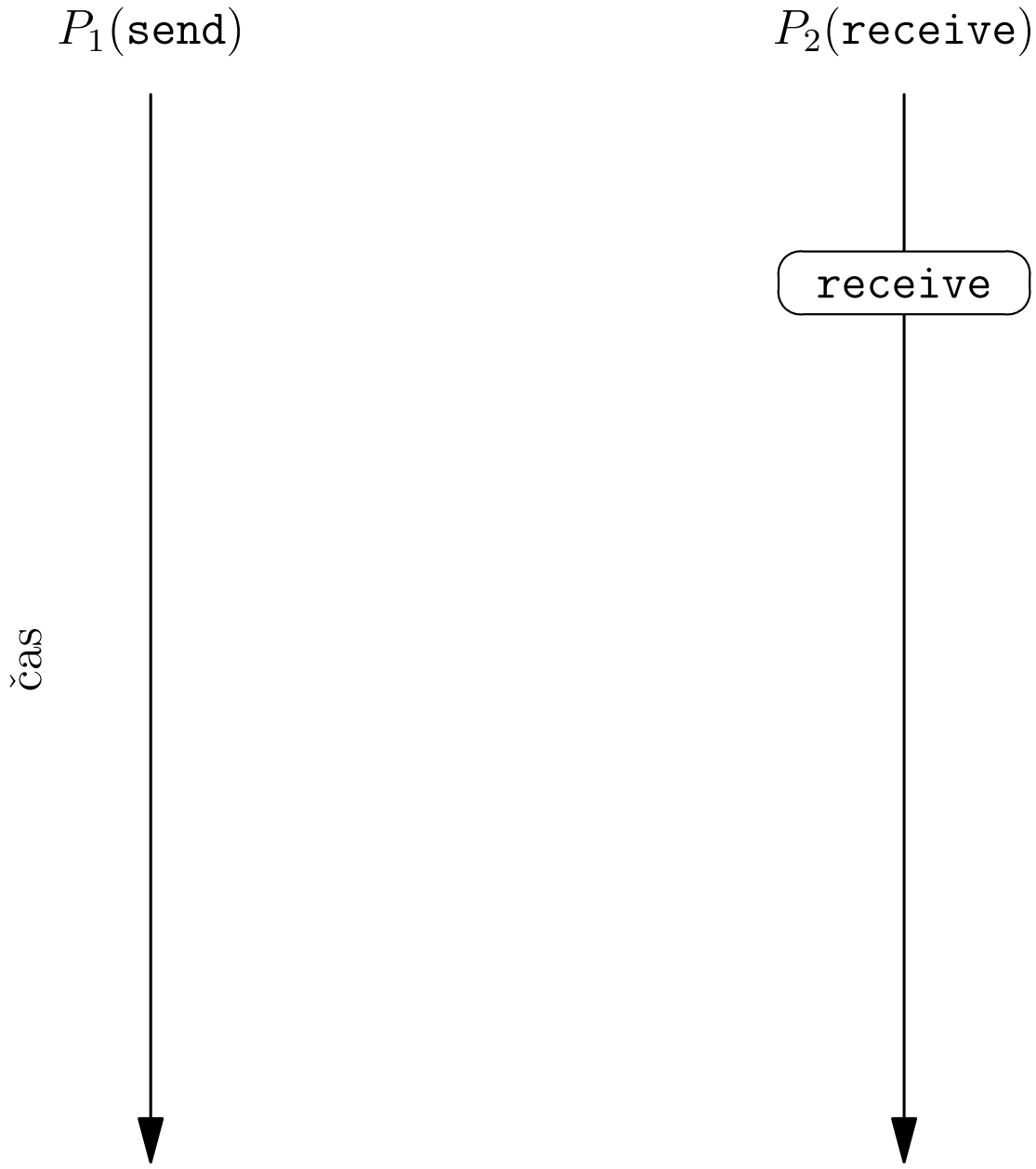
Blokující posílání zpráv bez bufferu



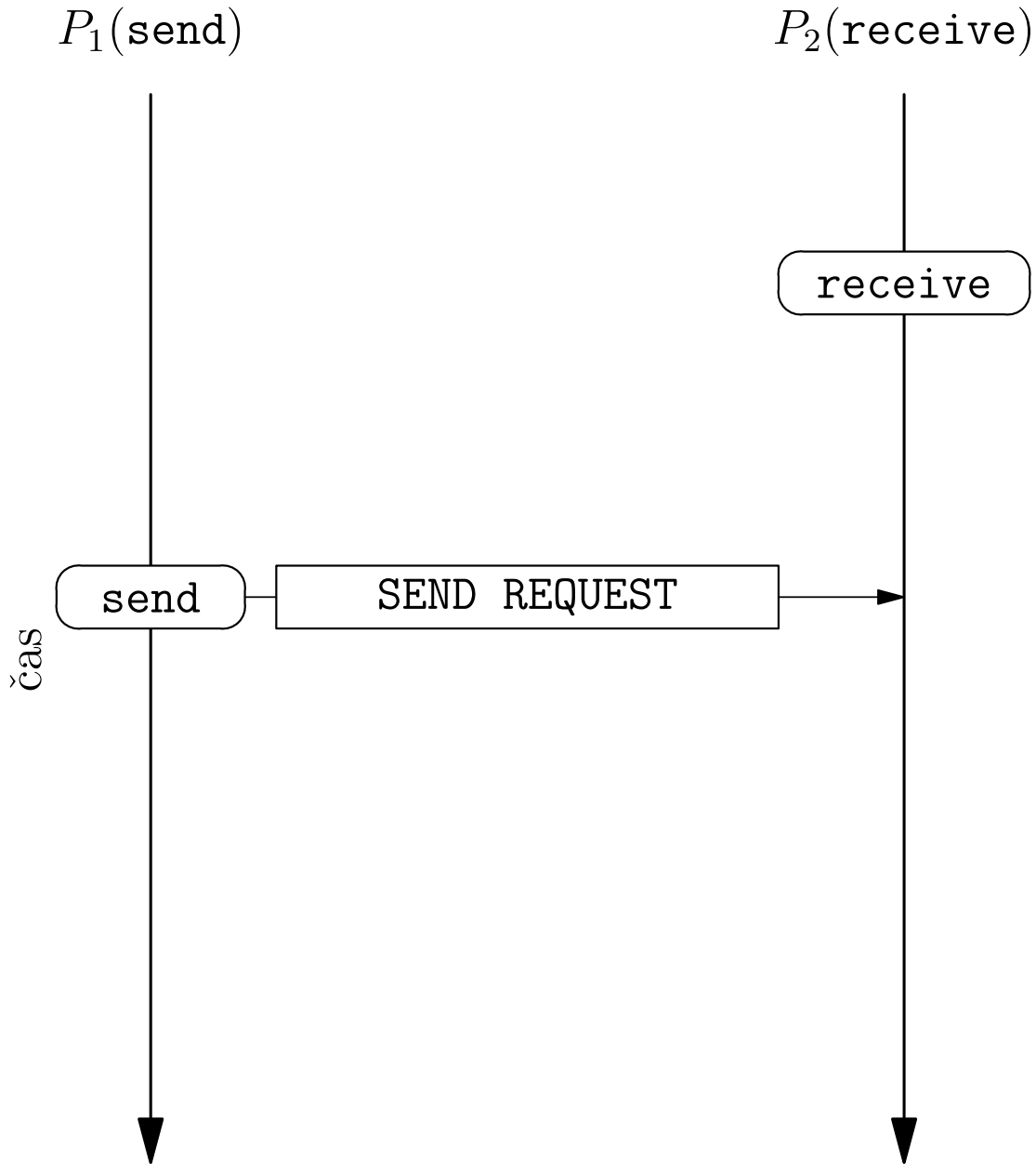
Blokující posílání zpráv bez bufferu



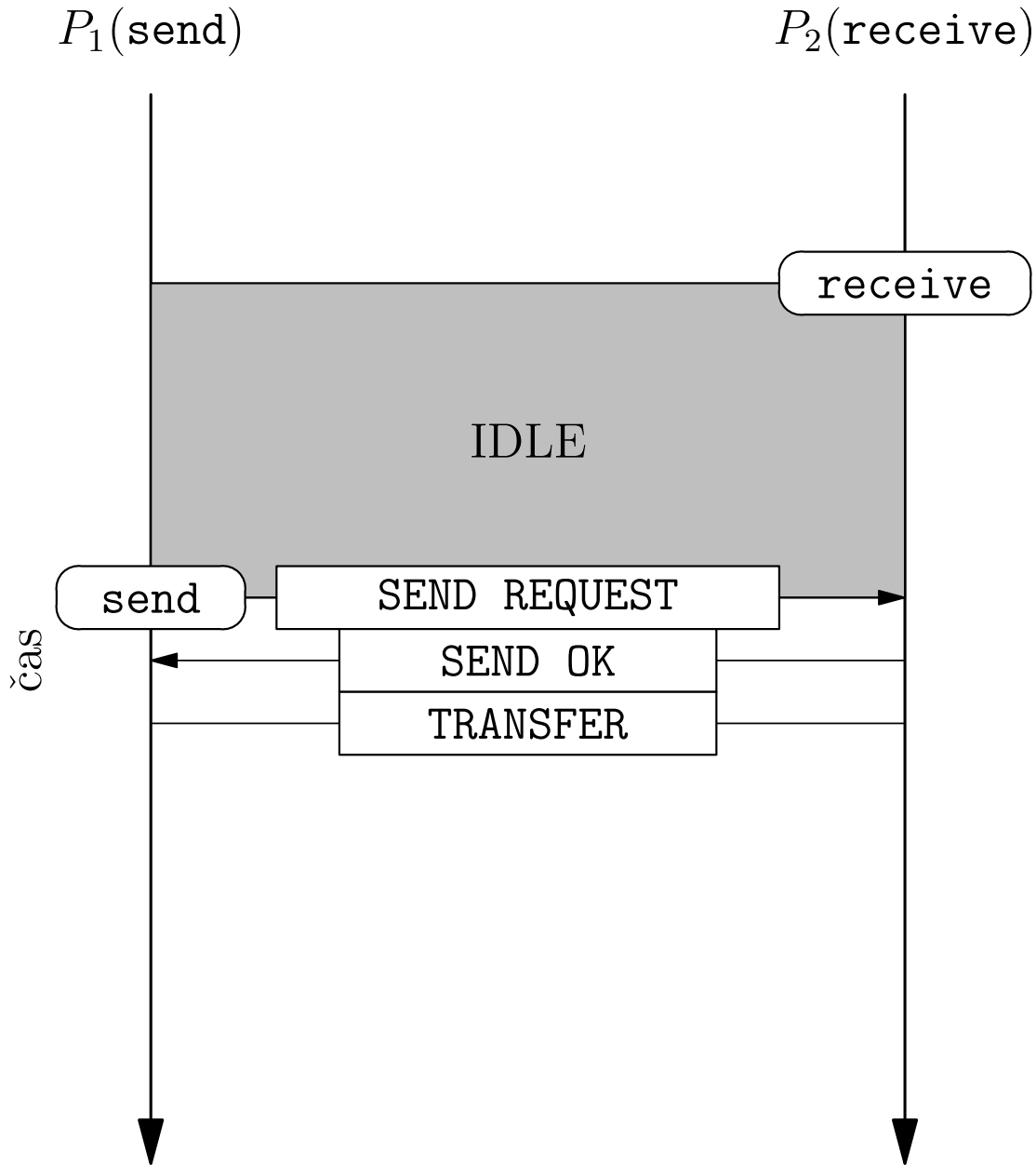
Blokující posílání zpráv bez bufferu



Blokující posílání zpráv bez bufferu



Blokující posílání zpráv bez bufferu



Blokující posílání zpráv bez bufferu

Možnost deadlocku:

P_1	P_2
`send(&a,1,2);`	`send( &a,1,1);`
`receive(&b,1,2);`	`receive( &b,1,1 );`

- oba procesy posílají *send request* a oba pak čekají na potvrzení *send ok*

Blokující posílání zpráv s bufferem

- snažíme se odstranit nutnost čekání na `SEND OK`
- při volání funkce `send` se data kopírují do pomocného bufferu
- program pak může okamžitě pokračovat
- přenos dat může proběhnout bez účasti CPU
- na straně příjemce se data také kopírují do pomocného bufferu
  - někdy je buffer jen na jedné straně

Blokující posílání zpráv s bufferem

- čekání odesílajícího procesu se podařilo zredukovat
- přibyla ale režie nutná při kopírování dat z/do bufferů
- pokud jsou procesy často synchronizovány, může být komunikace bez bufferů efektivnější
- pokud se buffery zaplní, dochází k čekání také
- příjem je vždy blokující
  - nesmí se pokračovat dál, dokud nejsou data zkopírována z bufferu příjemce

Blokující posílání zpráv s bufferem

Možnost deadlocku:

P_1	P_2
`receive(&b,1,2);`	`receive( &b,1,1 );`
`send(&a,1,2);`	`send( &a,1,1);`

- oba procesy čekají na příjem zprávy

Neblokující posílání zpráv

- komunikační funkce vrací řízení programu dříve, než je to sémanticky bezpečné
- programátor musí sám ohlídat, zda již byla data přenesena
- mezi tím ale může zpracovávat jiný kód, který napracuje s posílanými daty
- existují funkce, které řeknou, zda již komunikace úspěšně proběhla
- i neblokující komunikace může používat buffery

One-to-all broadcast/All-to-one reduction

One-to-all broadcast

- jeden proces má identická data o velikosti m a rozešle je všem ostatním
  - používá se např. při rozesílání konfiguračních parametrů

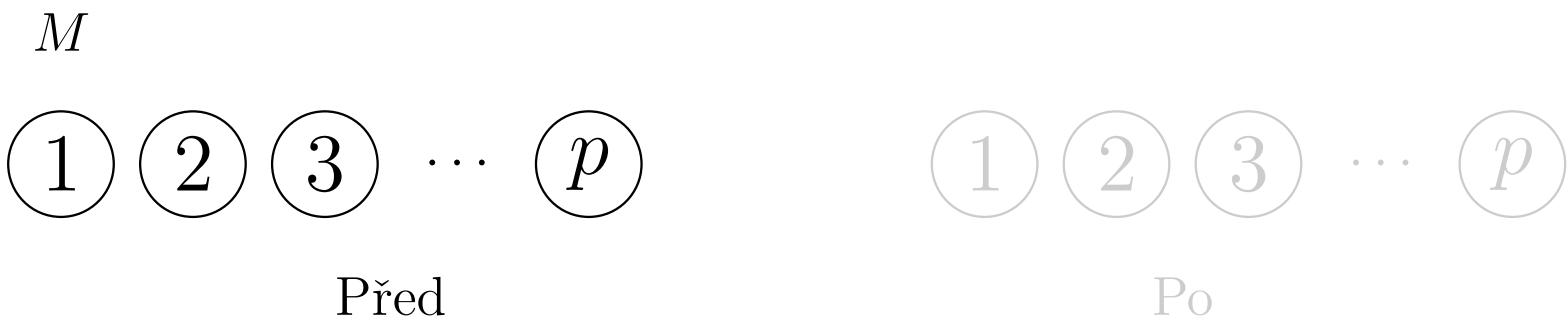
One-to-all broadcast/All-to-one reduction

All-to-one reduction

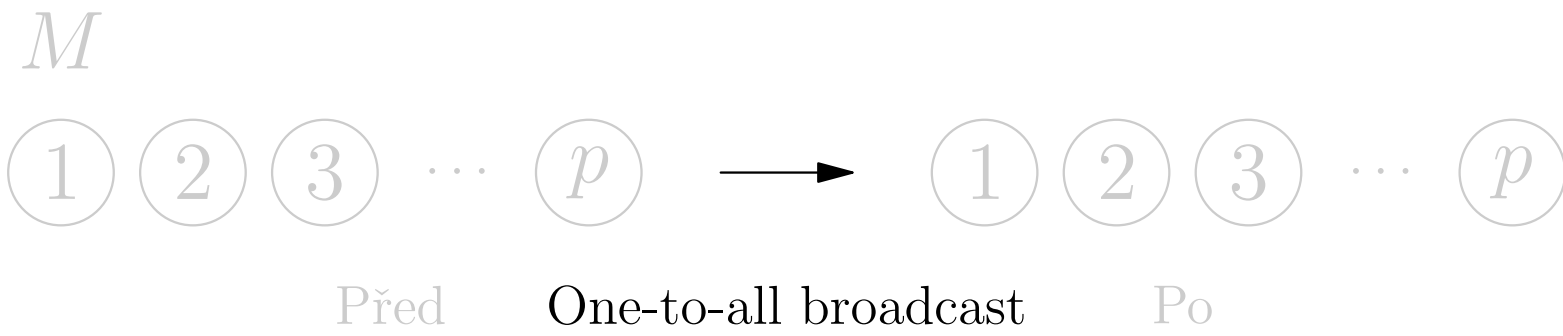
- každý proces má data o velikosti m a obecně jsou různá (různé hodnoty)
- data jsou poslána jednomu procesu a současně sloučena dohromady pomocí nějaké **asociativní operace** do velikosti m

Příklad: Máme 100 reálných čísel umístěných na 100 procesech (každý proces má jedno reálné číslo). Výsledkem redukce je součet všech čísel a výsledek má proces číslo 0.

One-to-all broadcast/All-to-one reduction

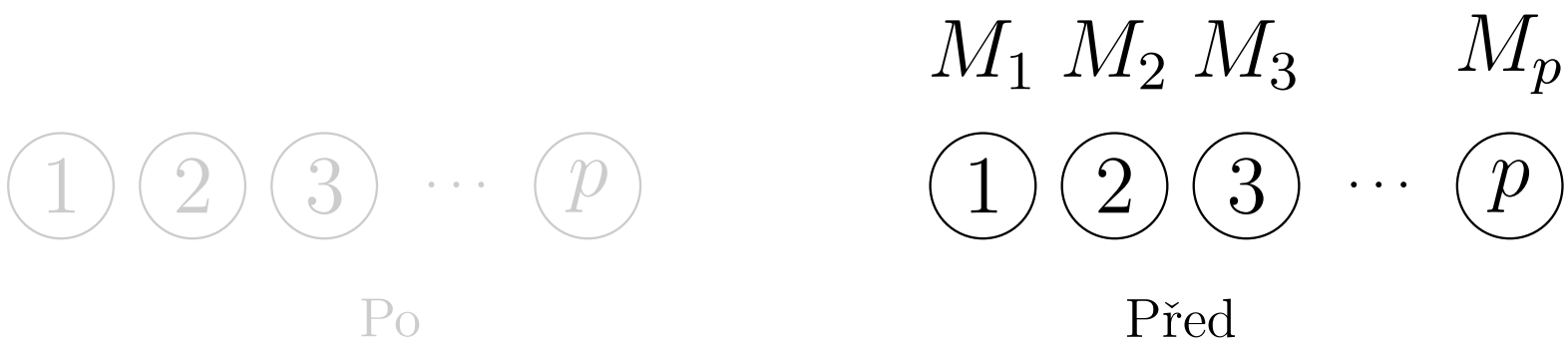


One-to-all broadcast/All-to-one reduction

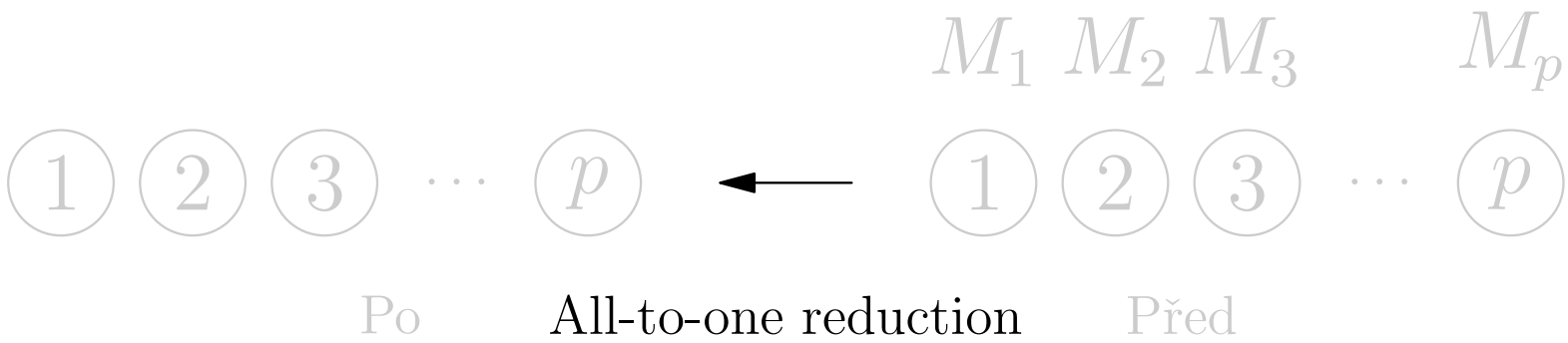


One-to-all broadcast/All-to-one reduction

One-to-all broadcast/All-to-one reduction



One-to-all broadcast/All-to-one reduction



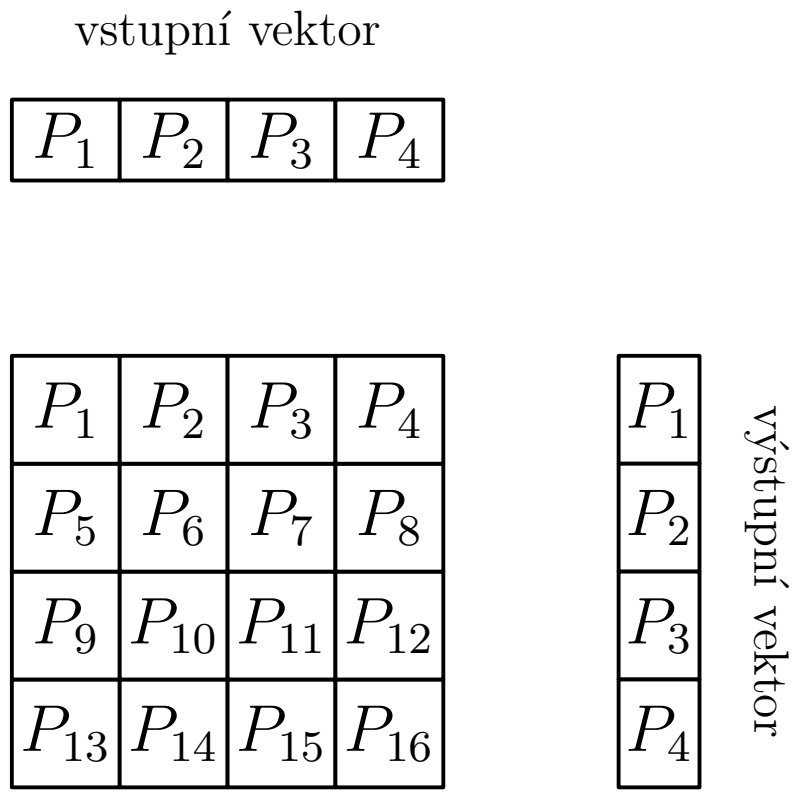
One-to-all broadcast/All-to-one reduction

$\sum_{i=1}^{p} M_i$

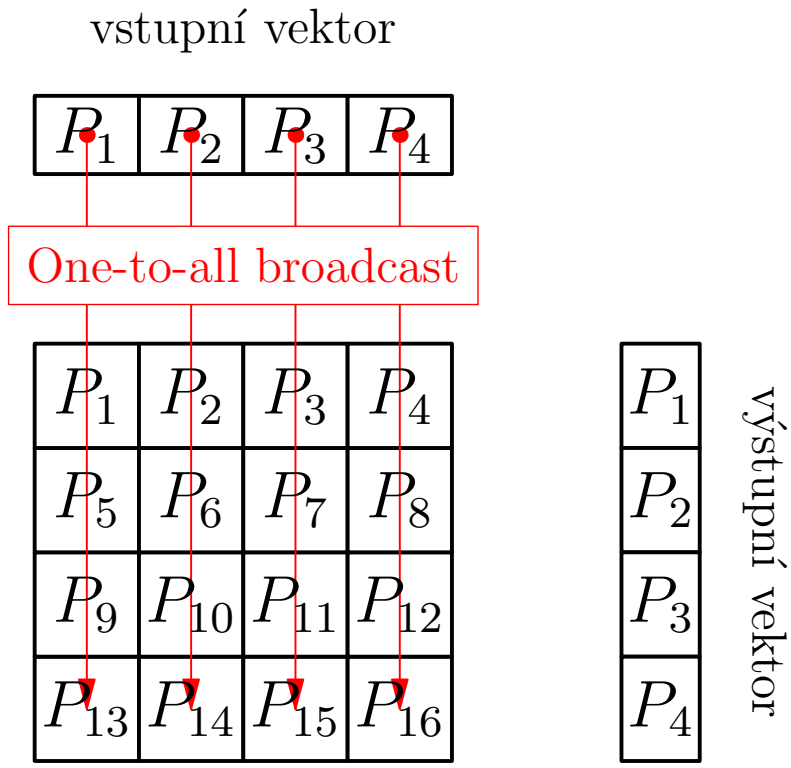
Po

Před

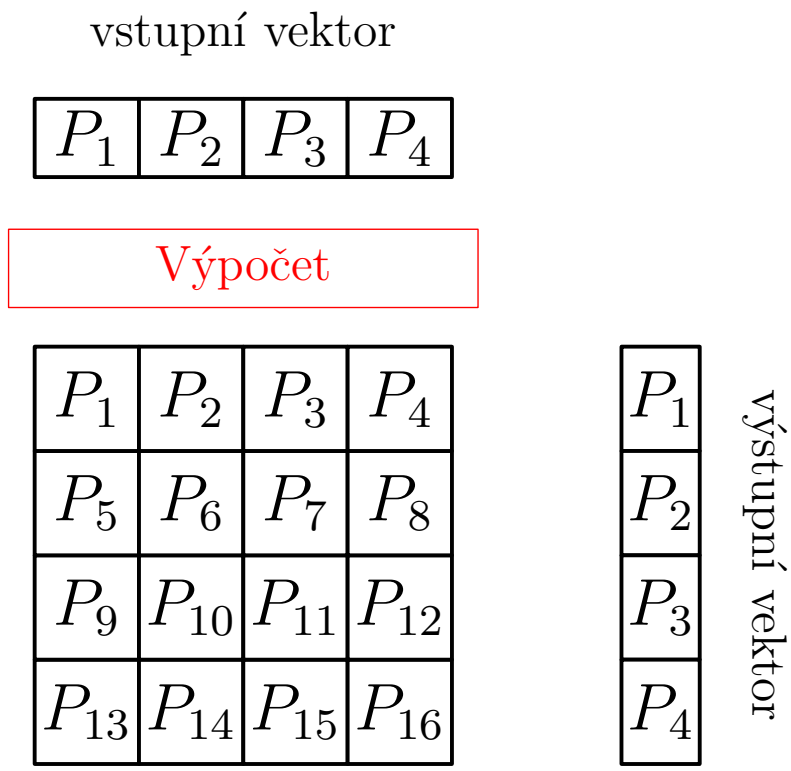
One-to-all broadcast/All-to-one reduction - příklad



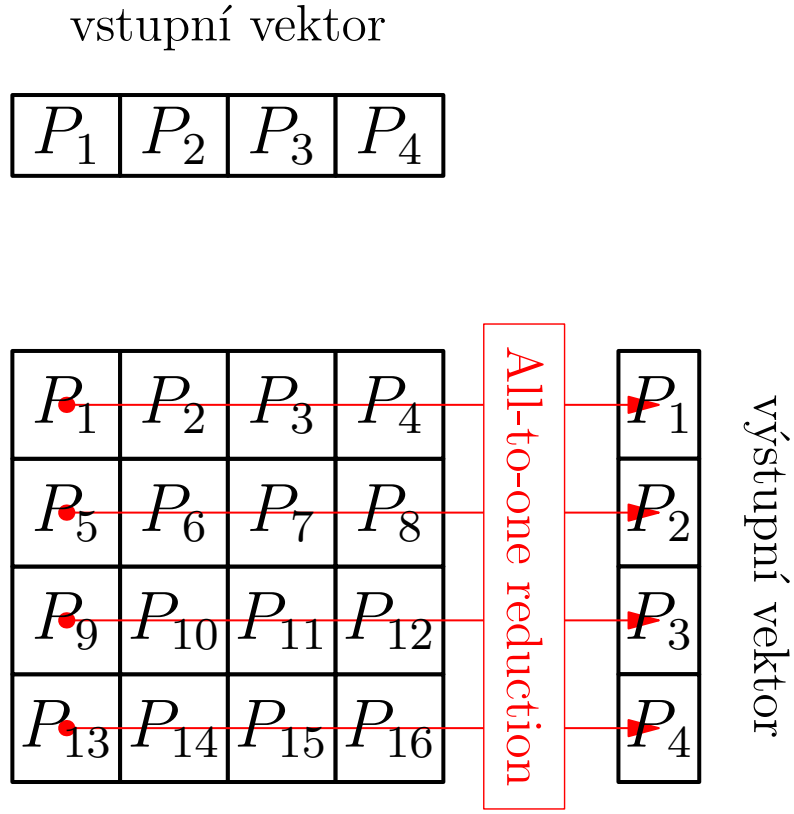
One-to-all broadcast/All-to-one reduction - příklad



One-to-all broadcast/All-to-one reduction - příklad

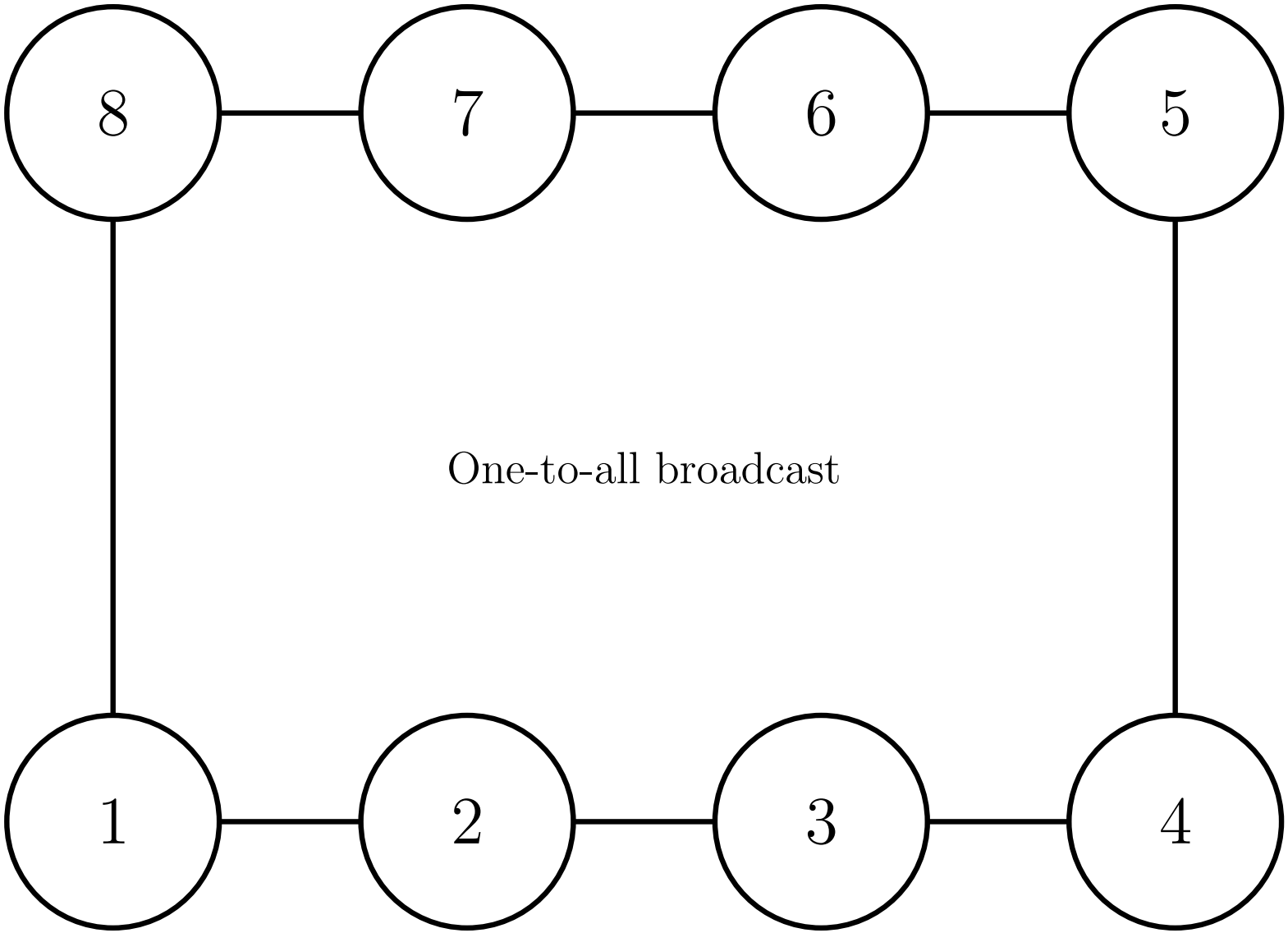


One-to-all broadcast/All-to-one reduction - příklad



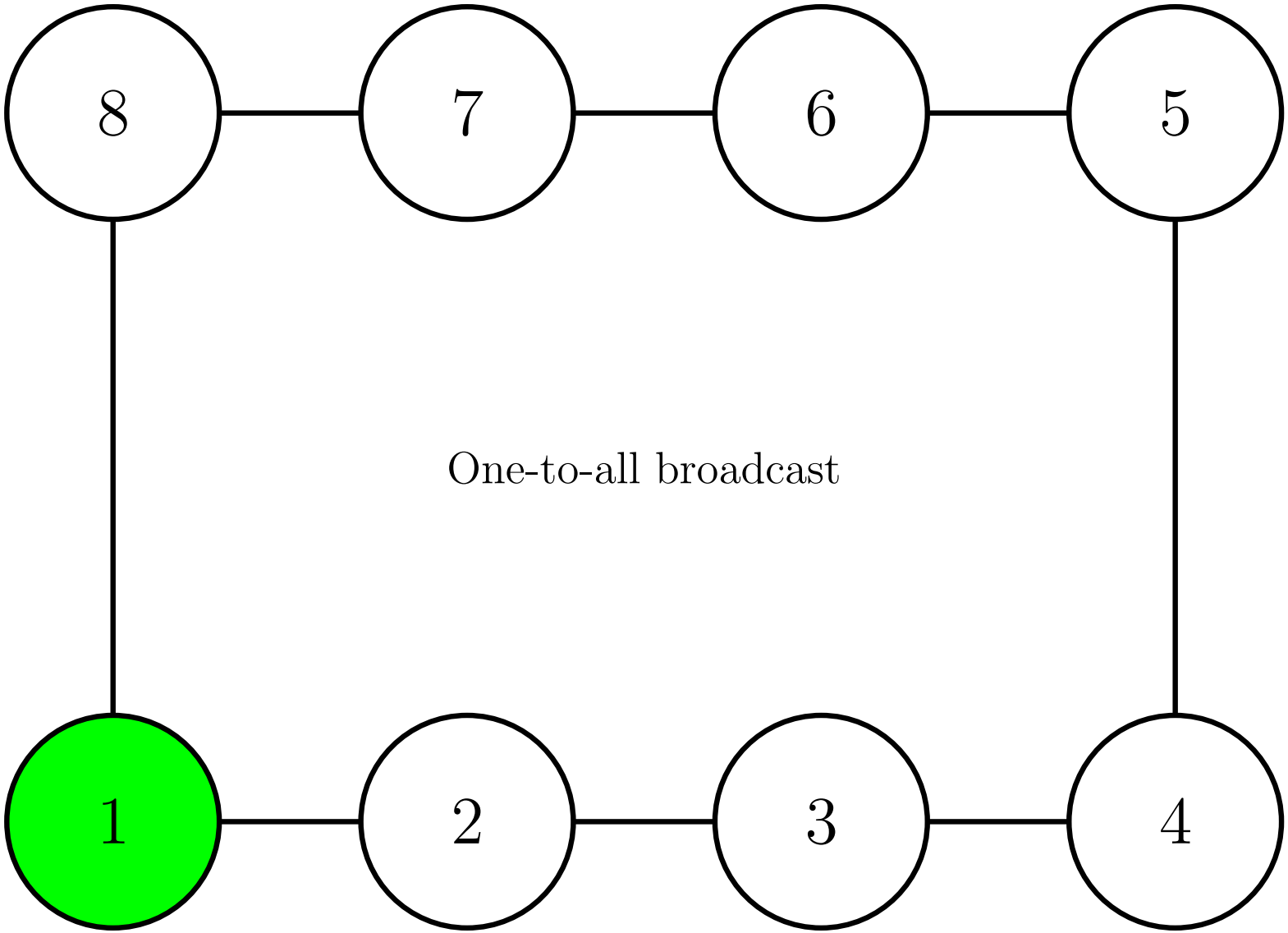
One-to-all broadcast/All-to-one reduction

Realizace v případě sítě typu **kruh**



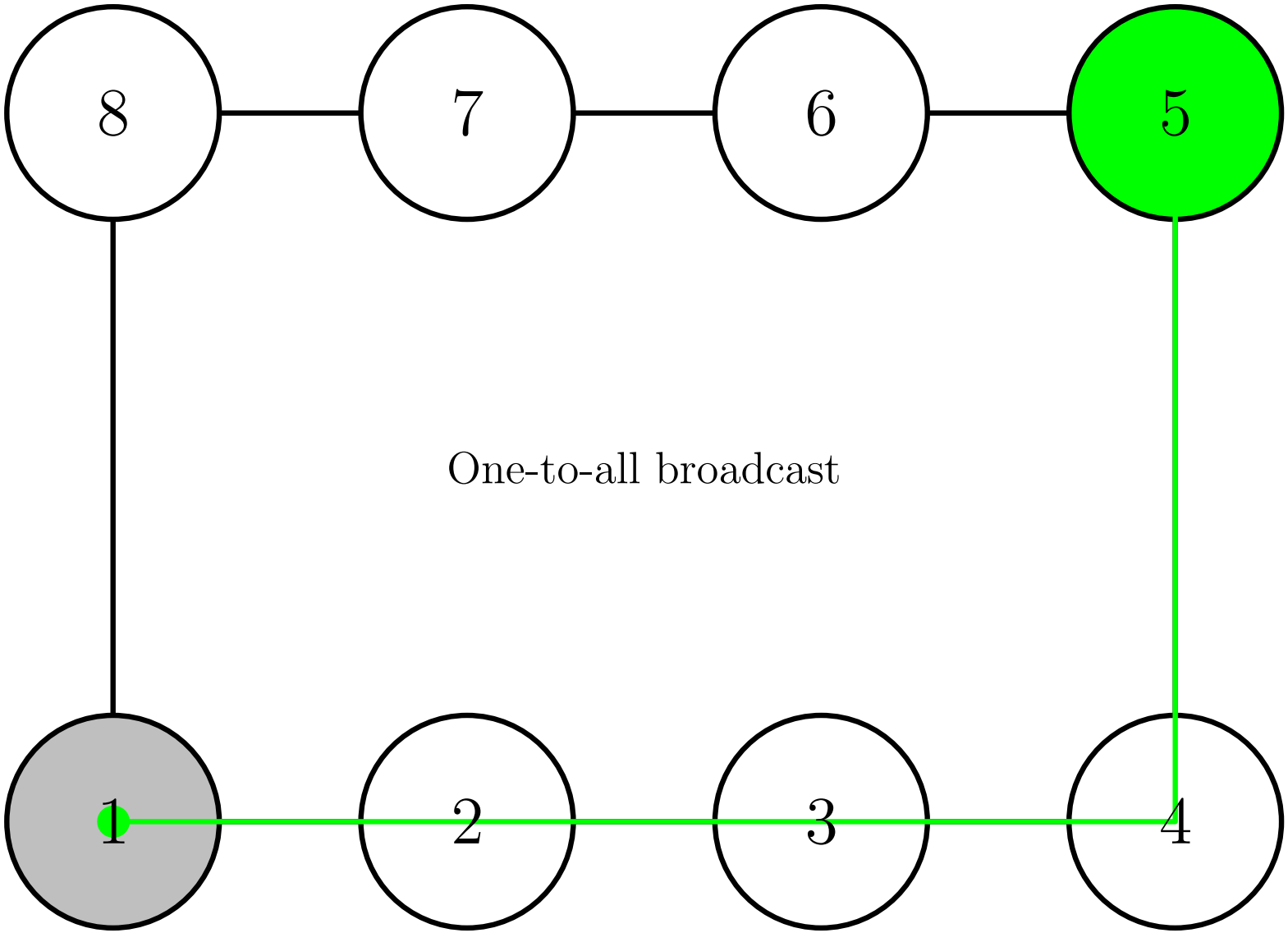
One-to-all broadcast/All-to-one reduction

Realizace v případě sítě typu **kruh**



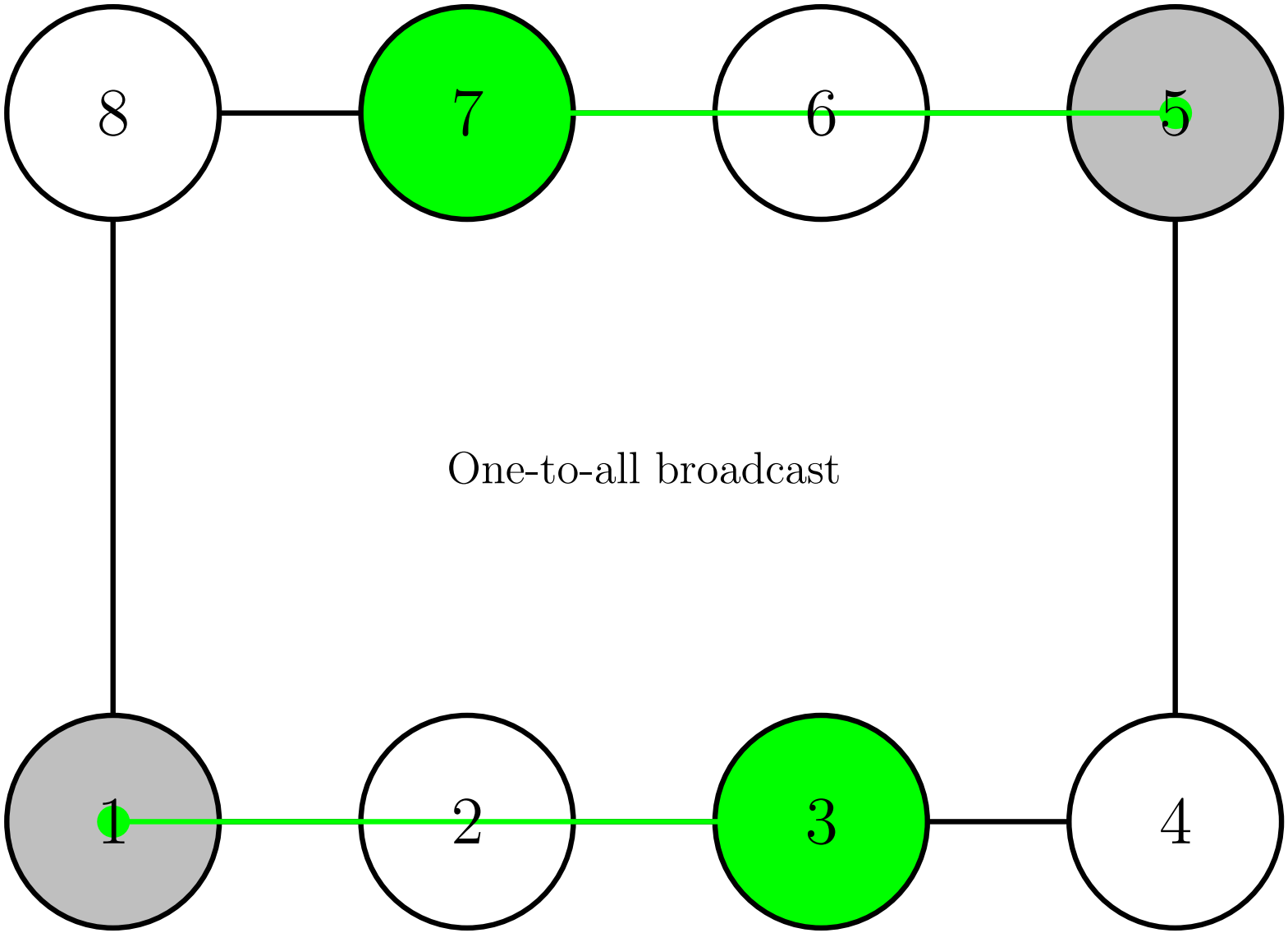
One-to-all broadcast/All-to-one reduction

Realizace v případě sítě typu **kruh**



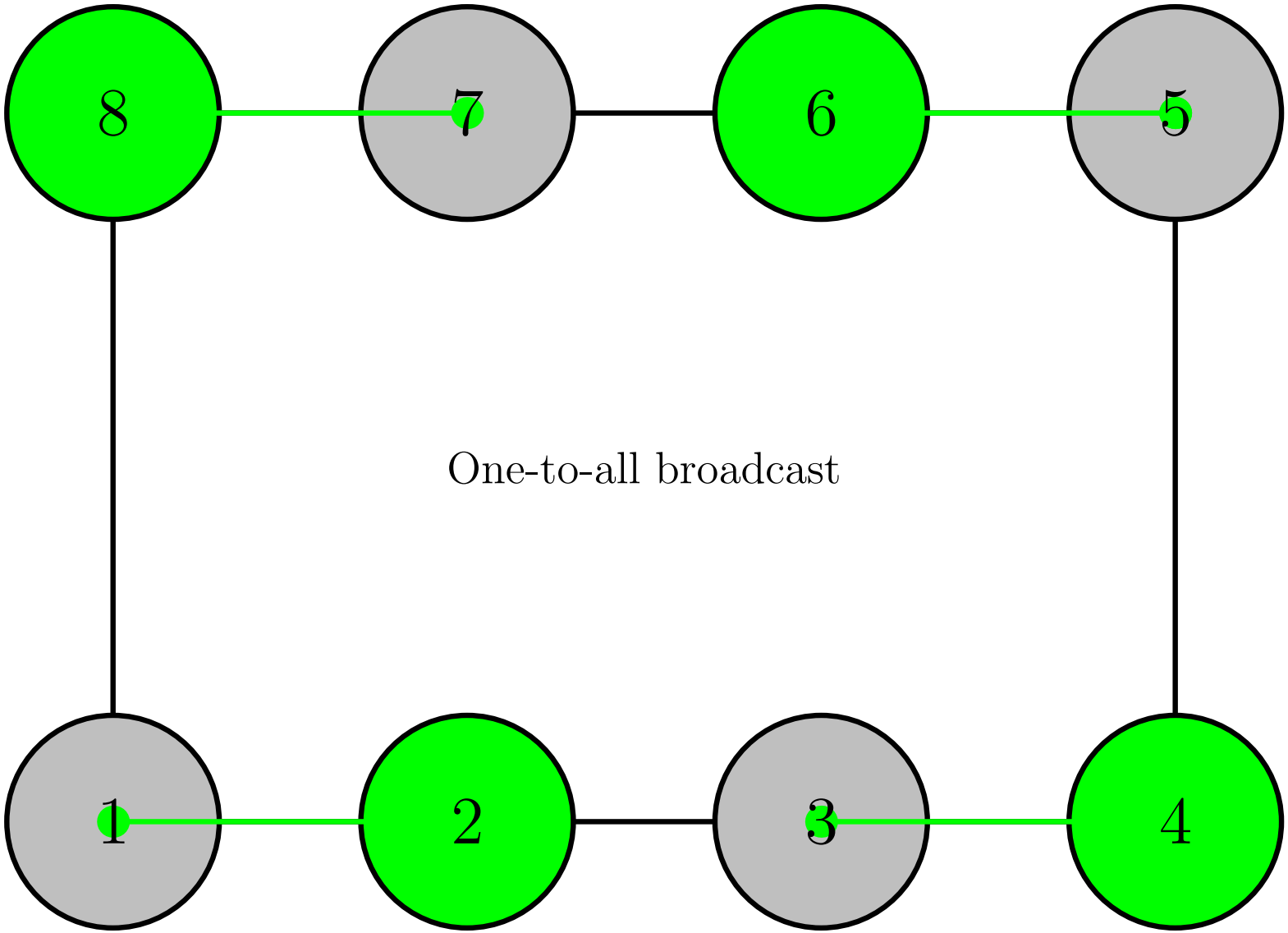
One-to-all broadcast/All-to-one reduction

Realizace v případě sítě typu **kruh**



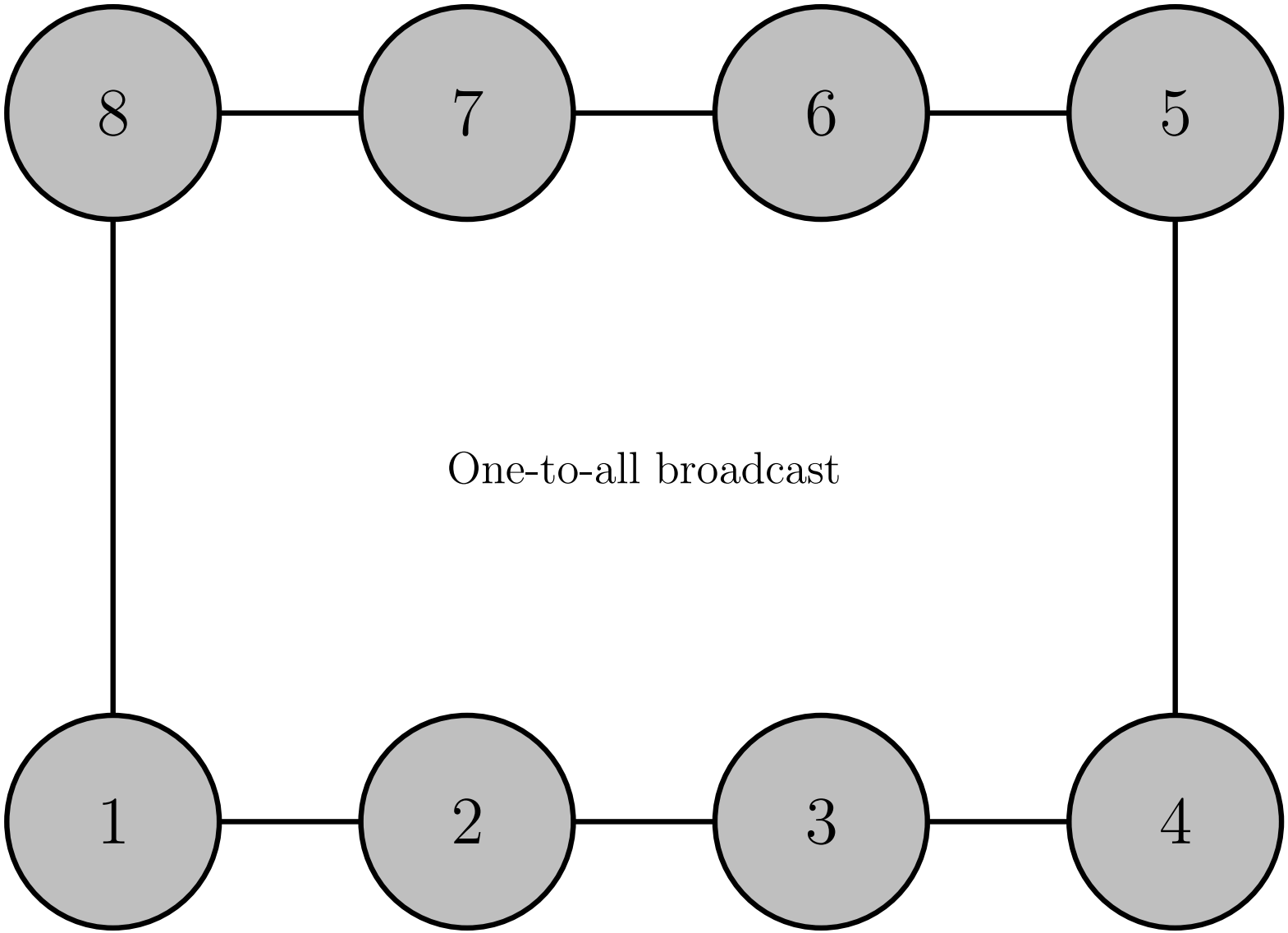
One-to-all broadcast/All-to-one reduction

Realizace v případě sítě typu **kruh**



One-to-all broadcast/All-to-one reduction

Realizace v případě sítě typu **kruh**



One-to-all broadcast/All-to-one reduction

Realizace v případě sítě typu **lineární řetězec** je stejná jako u kruhu.

- poslední spoj nebyl potřeba

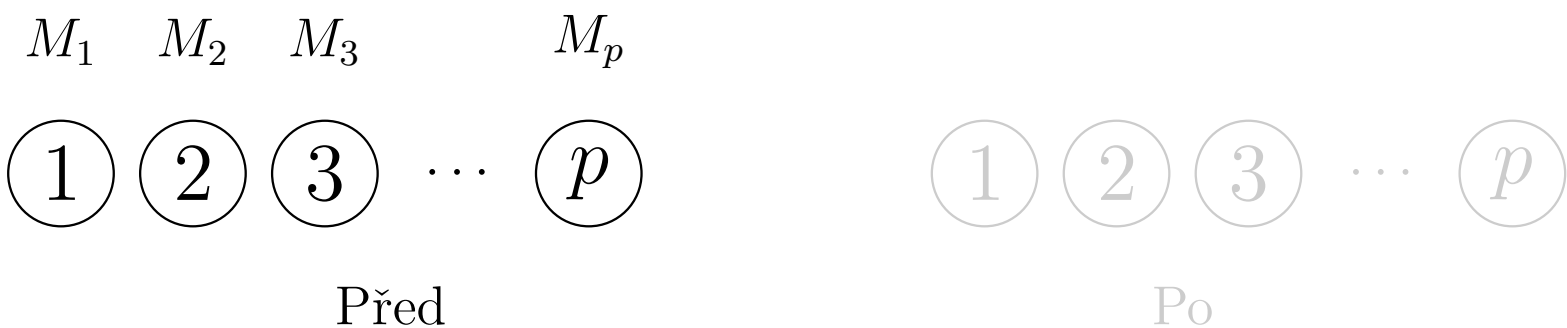
Realizace v případě **ortogonální sítě**:

- nejprve se provede *one-to-all broadcast* podél jedné souřadnice, následně podél další

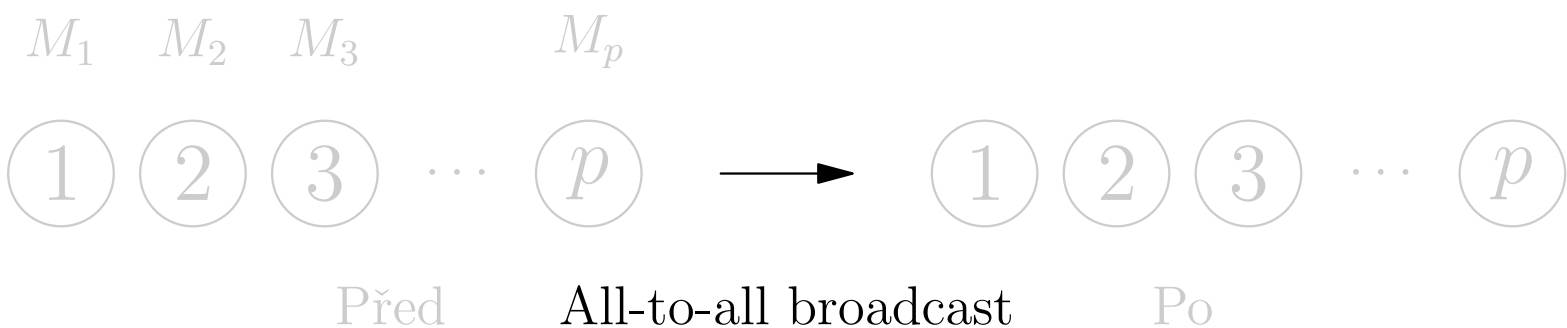
All-to-all broadcast/All-to-all reduction

Všech p procesů provádí současně *one-to-all broadcast* nebo *all-to-one reduction* s různými daty.

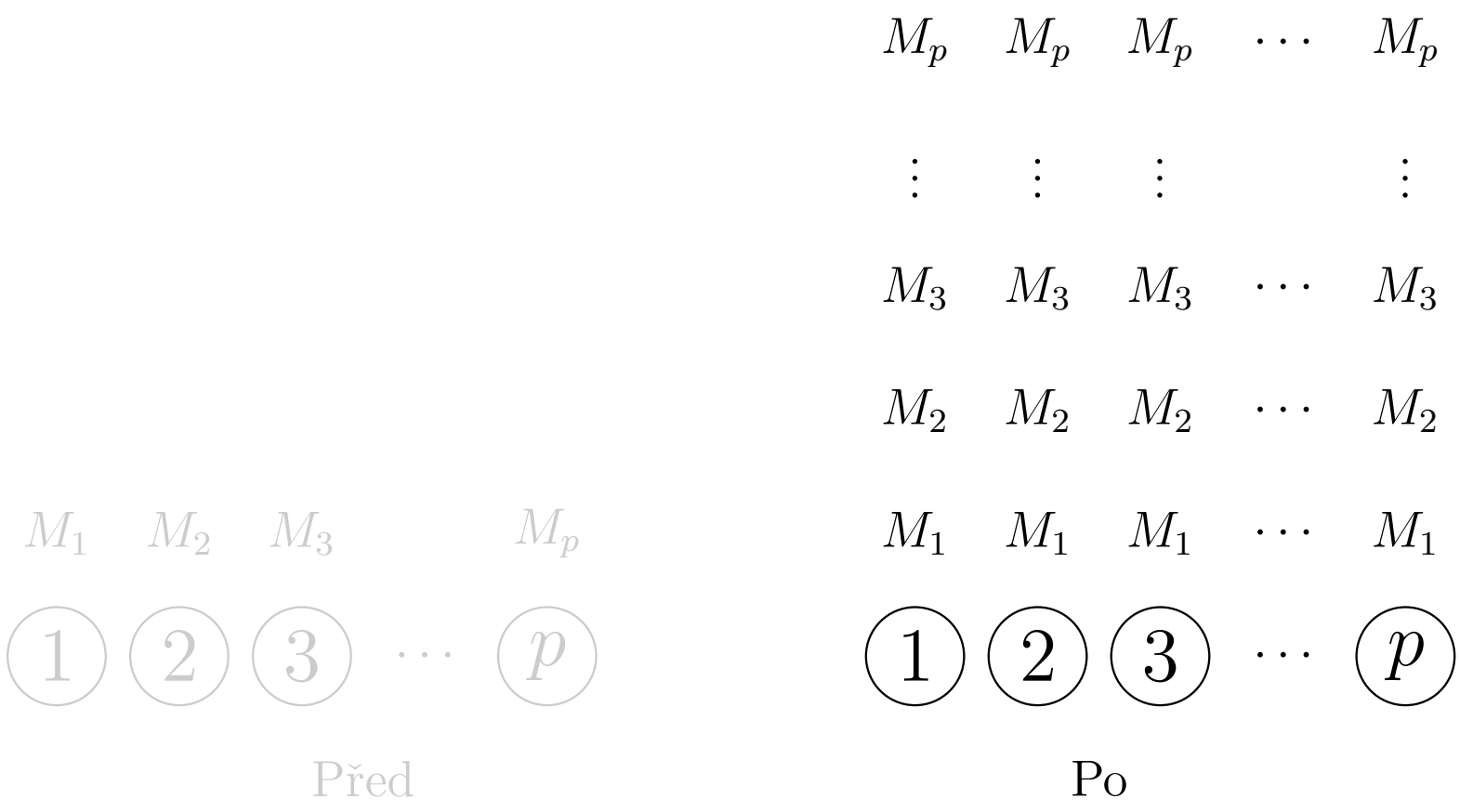
All-to-all broadcast/All-to-all reduction



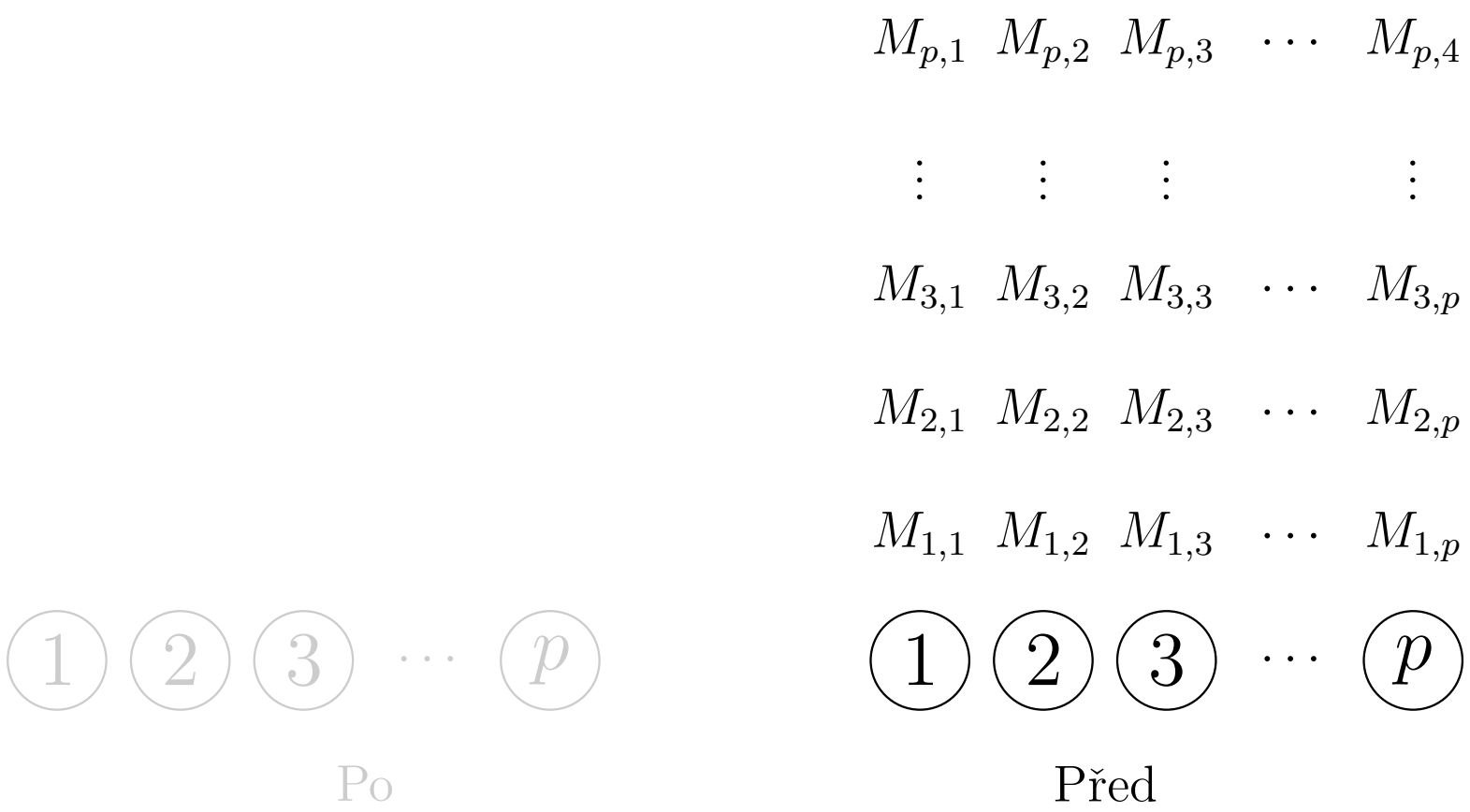
All-to-all broadcast/All-to-all reduction



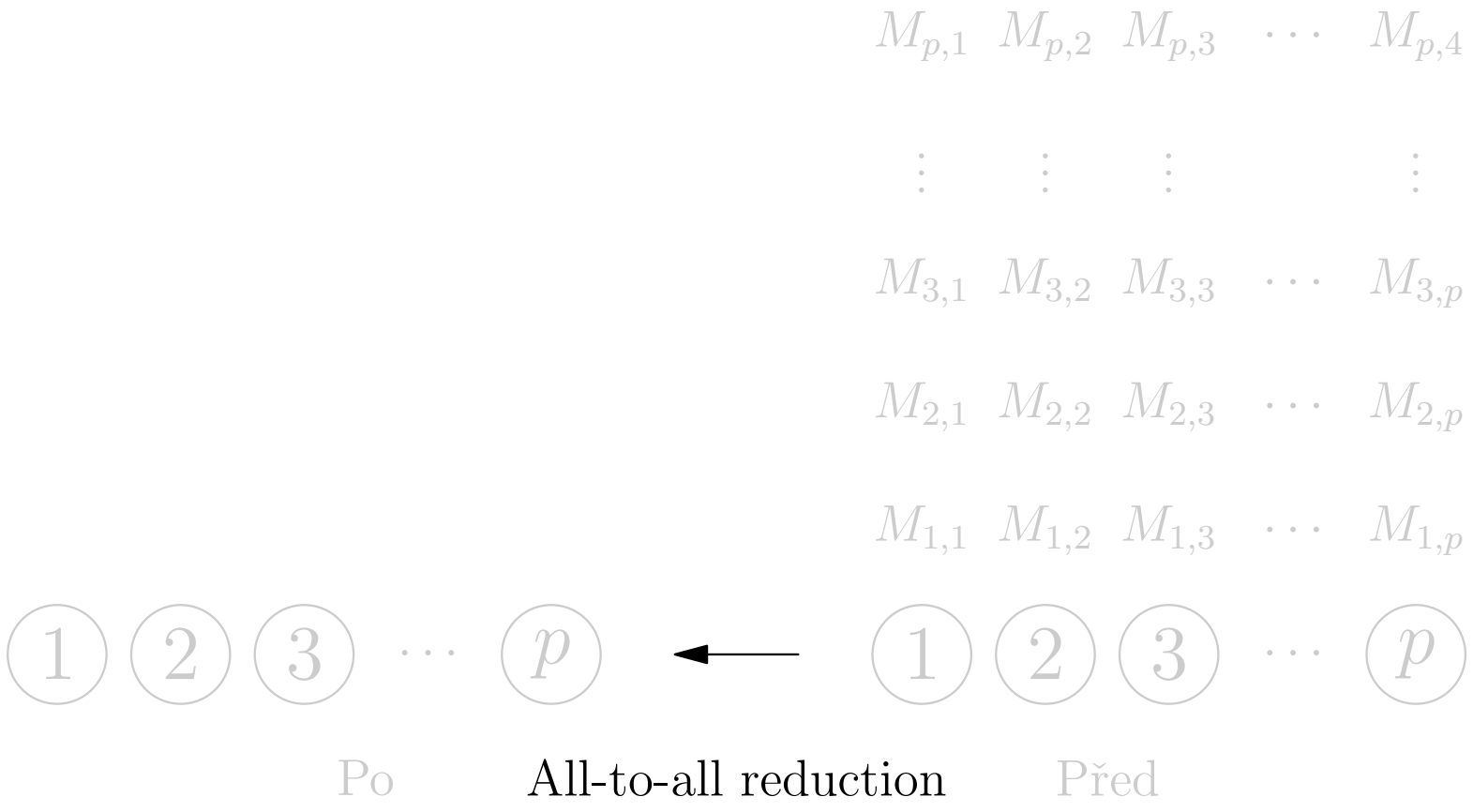
All-to-all broadcast/All-to-all reduction



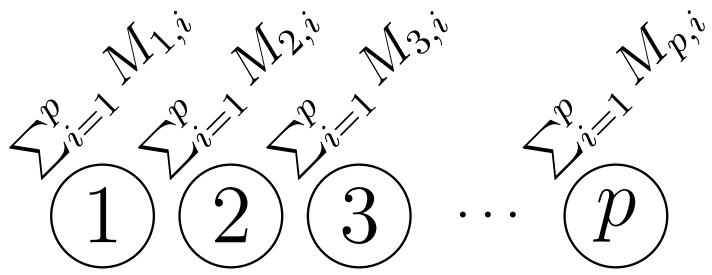
All-to-all broadcast/All-to-all reduction



All-to-all broadcast/All-to-all reduction



All-to-all broadcast/All-to-all reduction



Po

$$\begin{matrix} M_{p,1} & M_{p,2} & M_{p,3} & \cdots & M_{p,4} \\ \vdots & \vdots & \vdots & & \vdots \\ M_{3,1} & M_{3,2} & M_{3,3} & \cdots & M_{3,p} \\ M_{2,1} & M_{2,2} & M_{2,3} & \cdots & M_{2,p} \\ M_{1,1} & M_{1,2} & M_{1,3} & \cdots & M_{1,p} \\ 1 & 2 & 3 & \cdots & p \end{matrix}$$

Před

All-to-all broadcast/All-to-all reduction

Realizace *all-to-all broadcast* v případě sítě typu **kruh**:

- v každém kroku každý uzel pošle svoje data sousedovi vpravo a přijme data od souseda vlevo
- v následujícím kroku posílá data, která v předchozím kroku přijal
- vše se opakuje $p - 1$-krát

V případě **ortogonálních sítí** se opět postupuje podél jednotlivých souřadnic.

All-to-all broadcast/All-to-all reduction

Realizace *all-to-all reduction* v případě sítě typu **kruh**:

- v první kroku pošle uzel i data $M_{j,i}$ pro $j = 1, \cdots p$ a $j \neq i$ svému sousedovi vpravo a příjme data $M_{j,i-1}$ pro $j = 1, \cdots p$ a $j \neq i - 1$ od svého souseda vlevo
- v druhém kroku pošle uzel i data $M_{j,i-1}$ pro $j = 1, \cdots p$ a $j \neq i \wedge j \neq i - 1$, svému sousedovi vpravo a příjme data $M_{j,i-2}$ pro $j = 1, \cdots p$ a $j \neq i - 1 \wedge j \neq i - 2$ od svého souseda vlevo
- vše se ještě opakuje $p - 2$-krát za současného provádění redukční operace

All-reduce a Prefix-sum

All-reduce

- na počátku má každý proces různá data M_i o velikosti m
- na konci mají všechny procesy stejný "součet"dat $\sum_{i=1}^{p} M_i$
- All-reduce lze provést jako
  - All-to-one reduction + One-to-all broadcast
  - All-to-all broadcast, kde se ale provede součet $\sum_{i=1}^{p} M_i$
- pro $m = 1$ lze All-reduce použít jako bariéru [1]
  - redukci nelze dokončit, dokud k ní nepřispěje každý proces svým podílem

[1] **Bariéra** je místo v programu, které nesmí žádný proces překročit, dokud ho nedosáhnou všechny ostatní procesy. Má význam synchronizace procesů.

All-reduce a Prefix-sum

Prefix-sum

- jde o modifikaci all-reduce
- pro data $M_1, \cdots M_p$ chceme najít $S_k = \sum_{i=1}^{k} M_i$ pro $k = 1, \cdots p$
- postup je stejný jako u all-reduce, ale každý proces k si "přičítá"jen data od procesů s ID $< k$
- některá komunikace je tu tady zbytečná, ale celkovou složitost této operace to neovlivní

Scatter/Gather

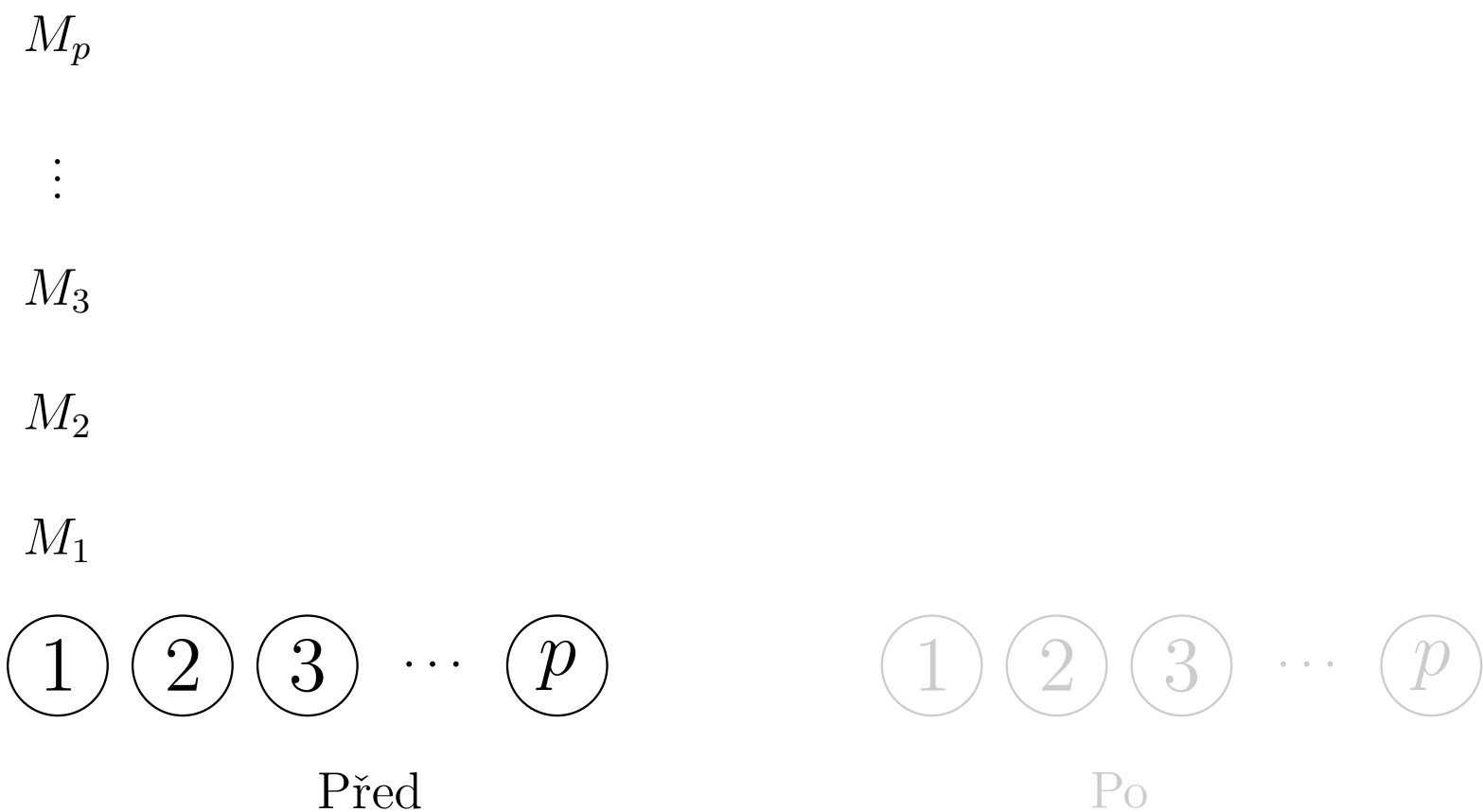
Scatter

- jeden proces má na počátku p různých zpráv $M_1, \cdots M_p$
- proces i má na konci zprávu M_i
- někdy se tato zpráva nazývá **one-to-all personalized communicatio**

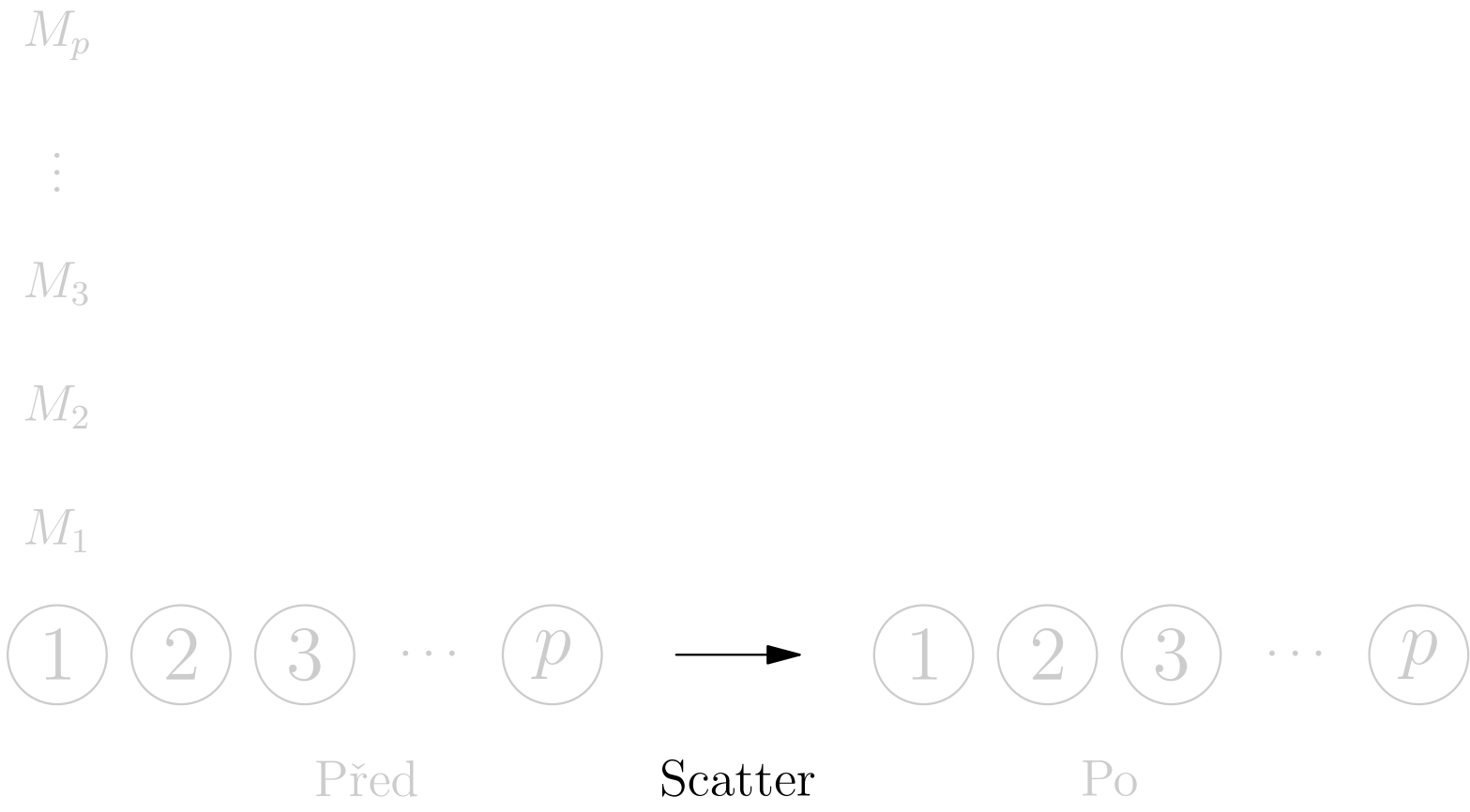
Gather

- je to duální operace ke *scatter*
- na počátku má každý proces i zprávu M_i
- na konci má jeden proces všechny zprávy $\cap_{i=1}^{p} M_i$

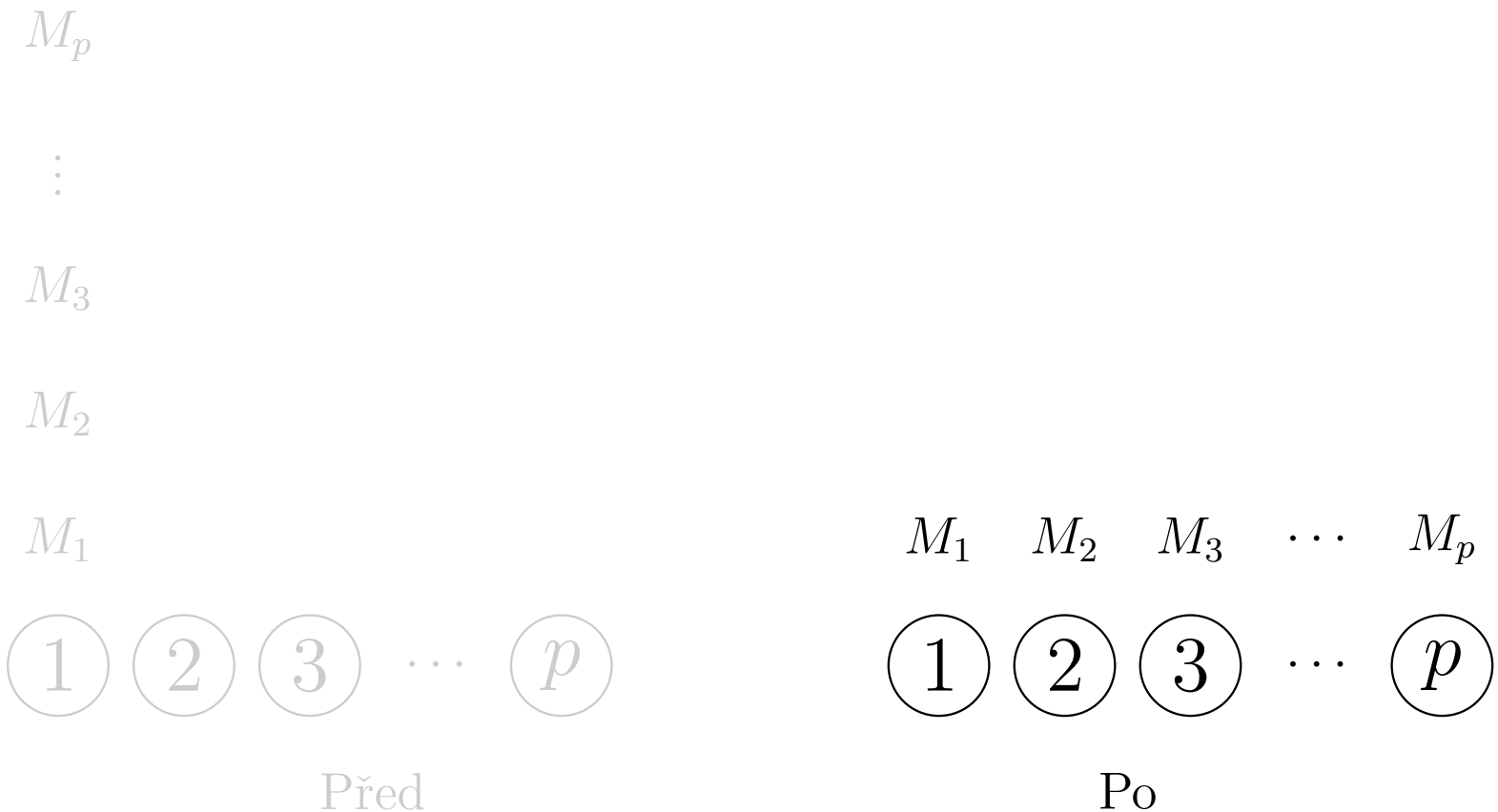
Scatter/Gather



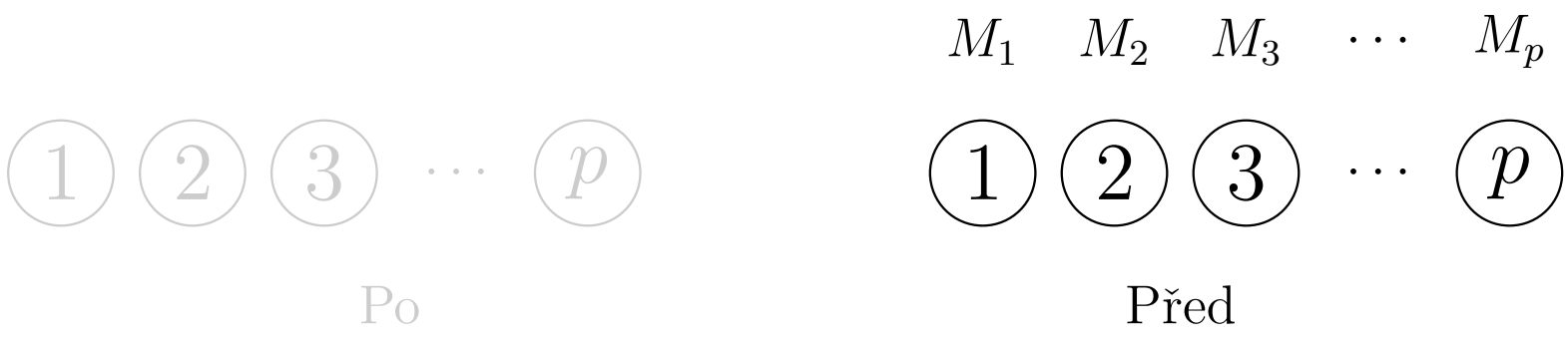
Scatter/Gather



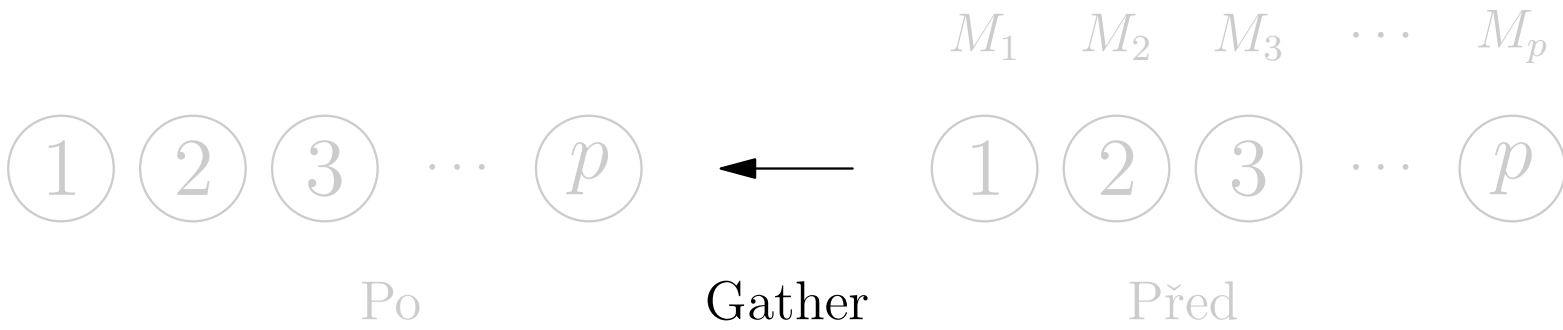
Scatter/Gather



Scatter/Gather



Scatter/Gather



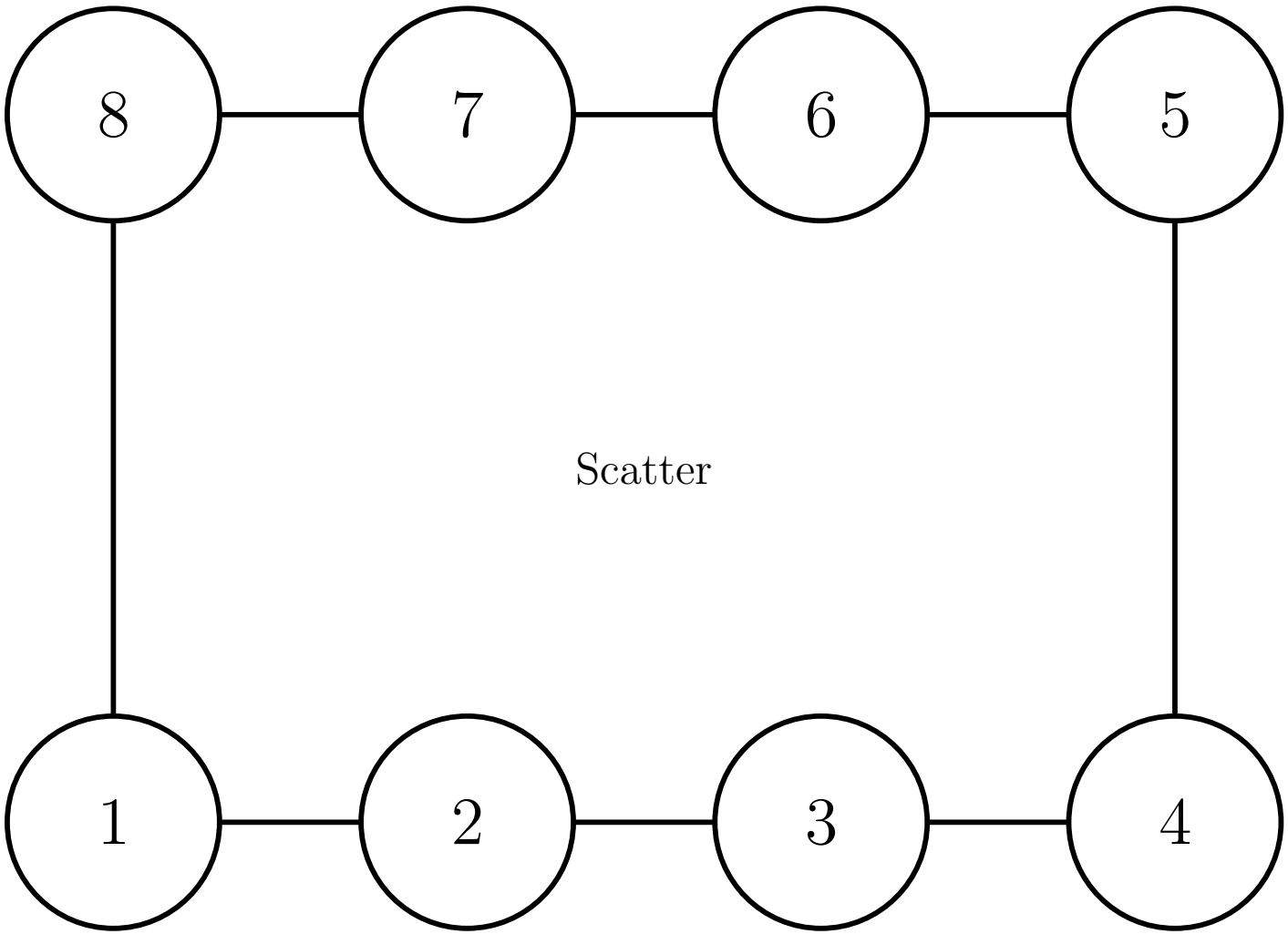
Scatter/Gather

Scatter/gather

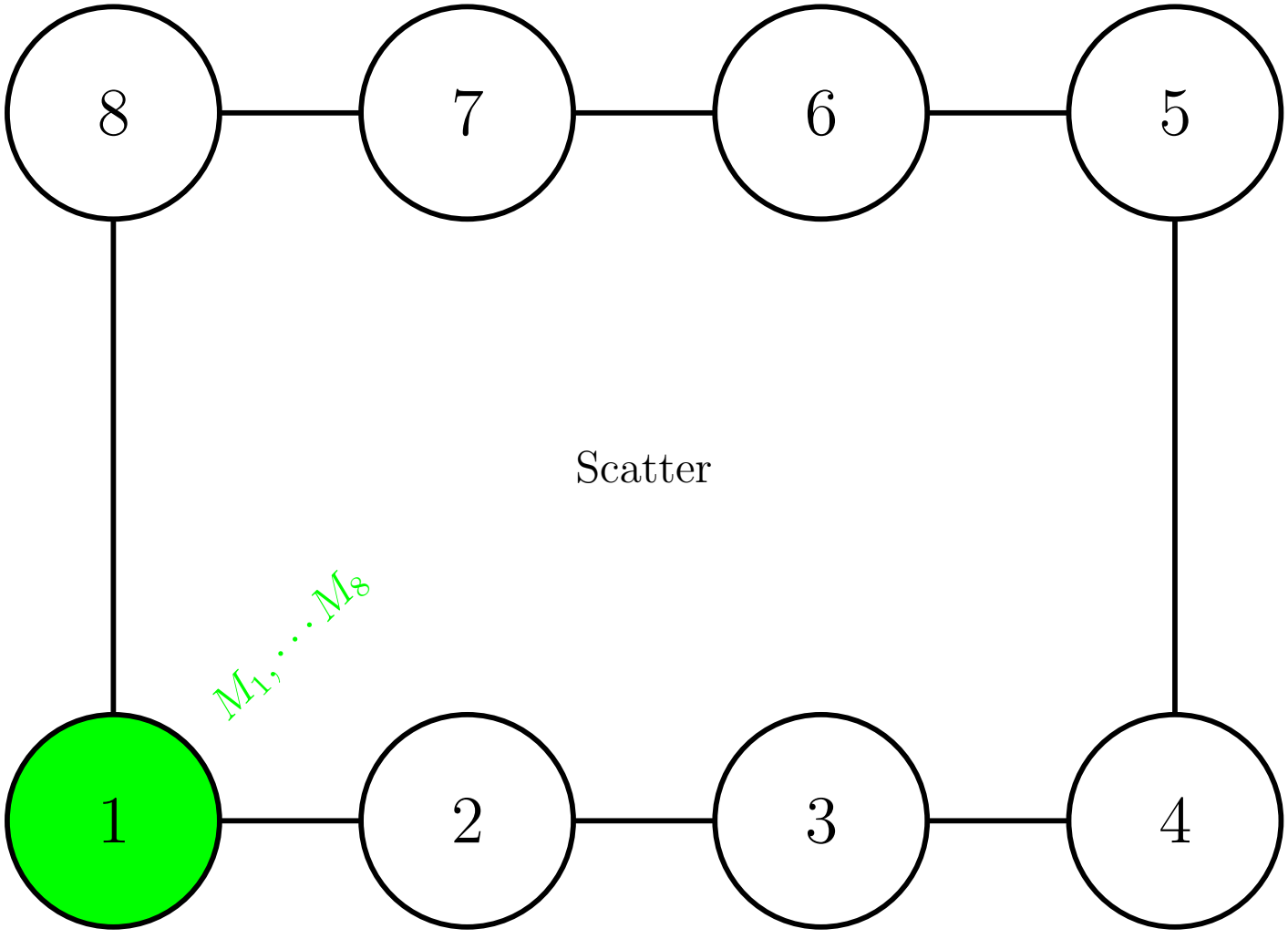
Realizace *scatter* v případě sítě typu **kruh**:

- je stejná jako u *one-to-all broadcast* ale s jinými objemy dat

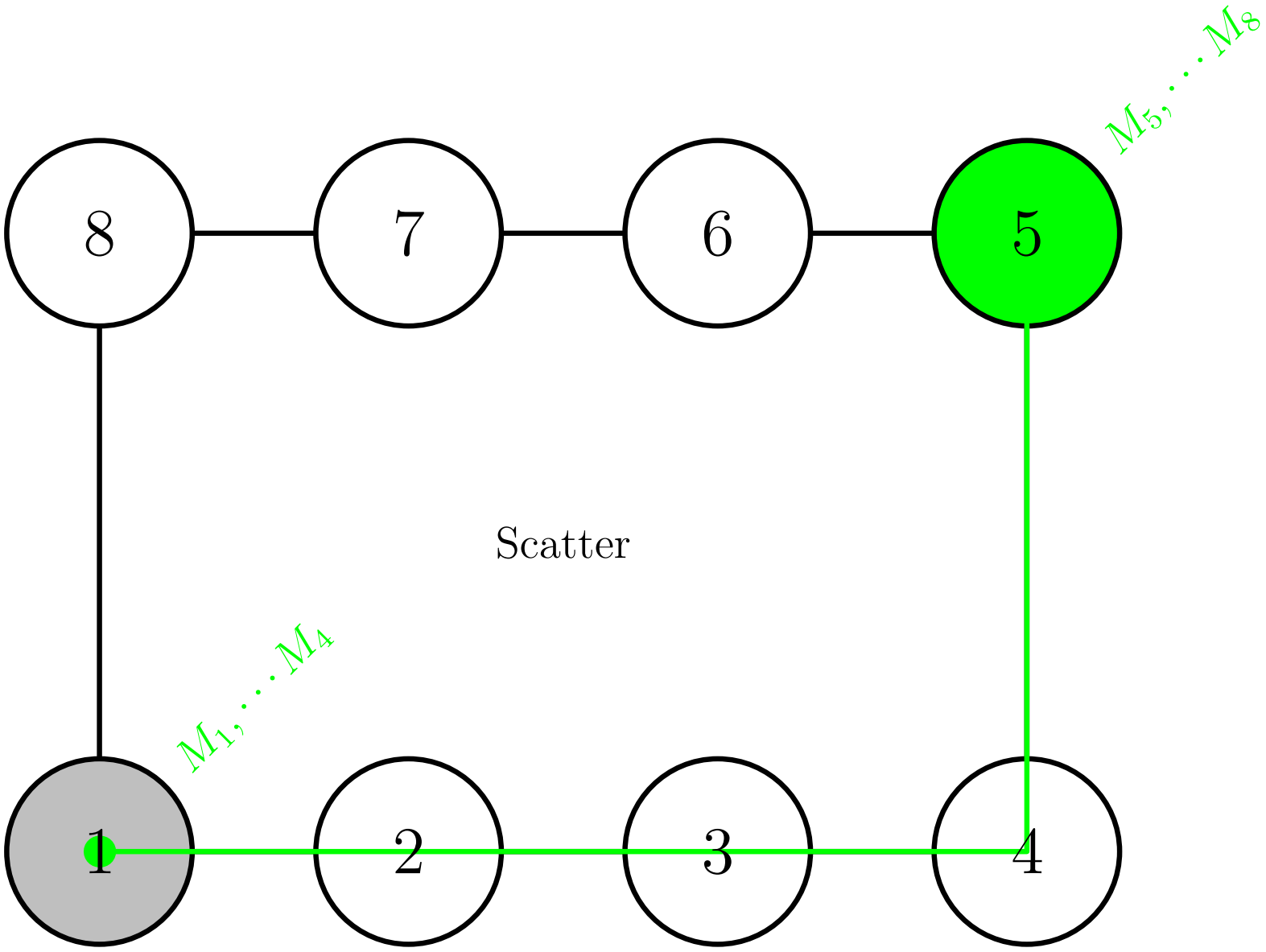
Scatter/gather



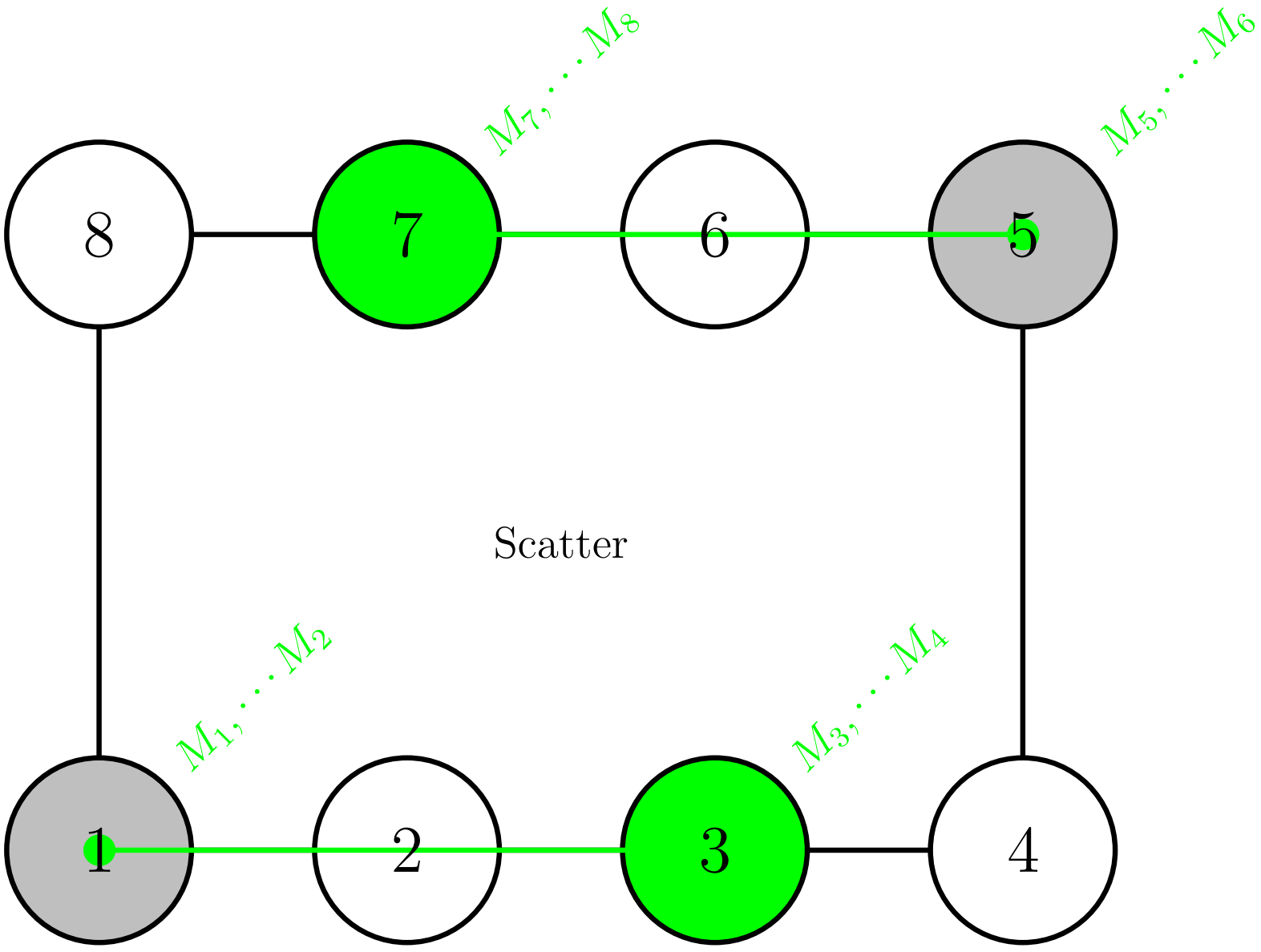
Scatter/gather



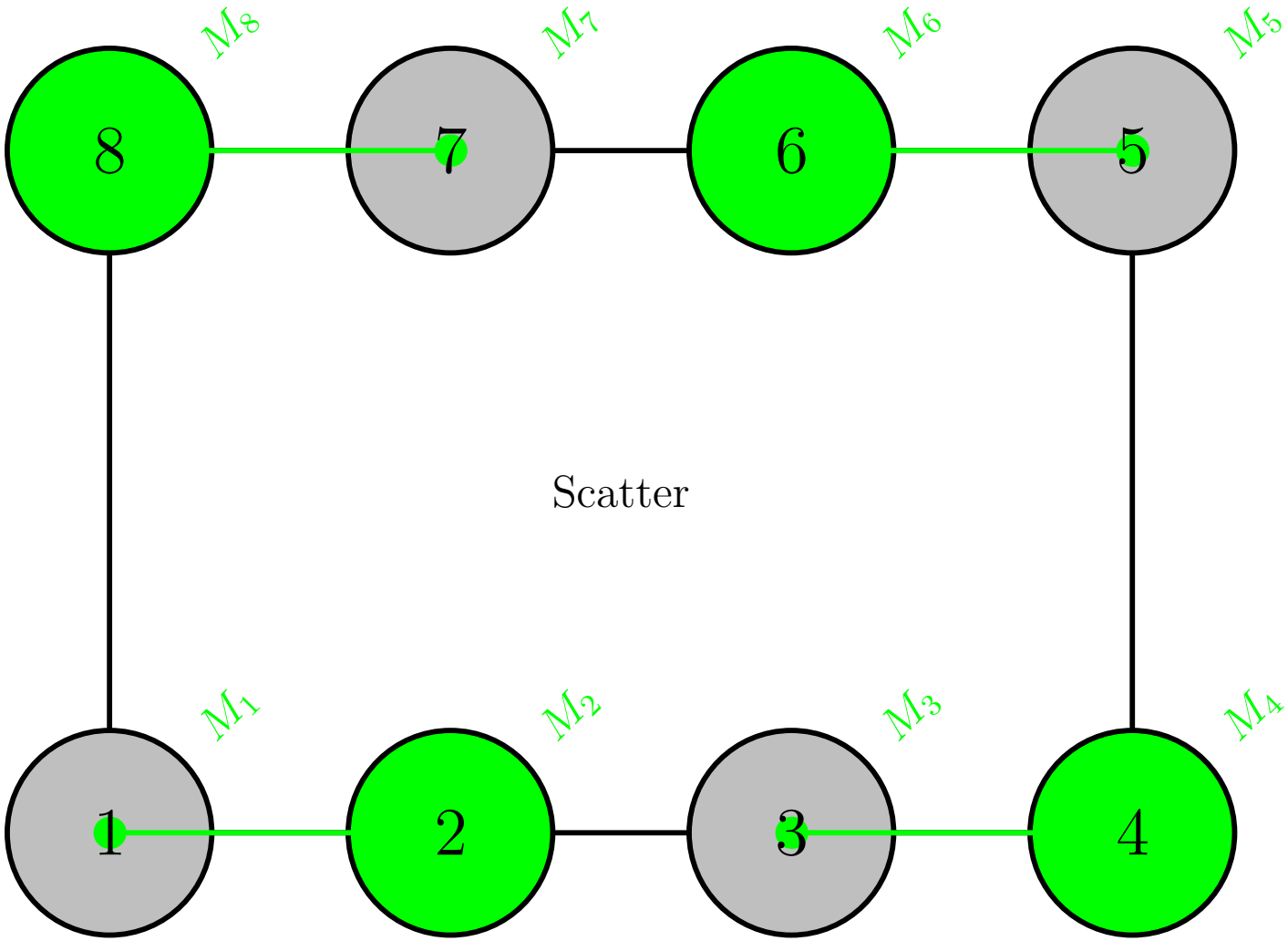
Scatter/gather



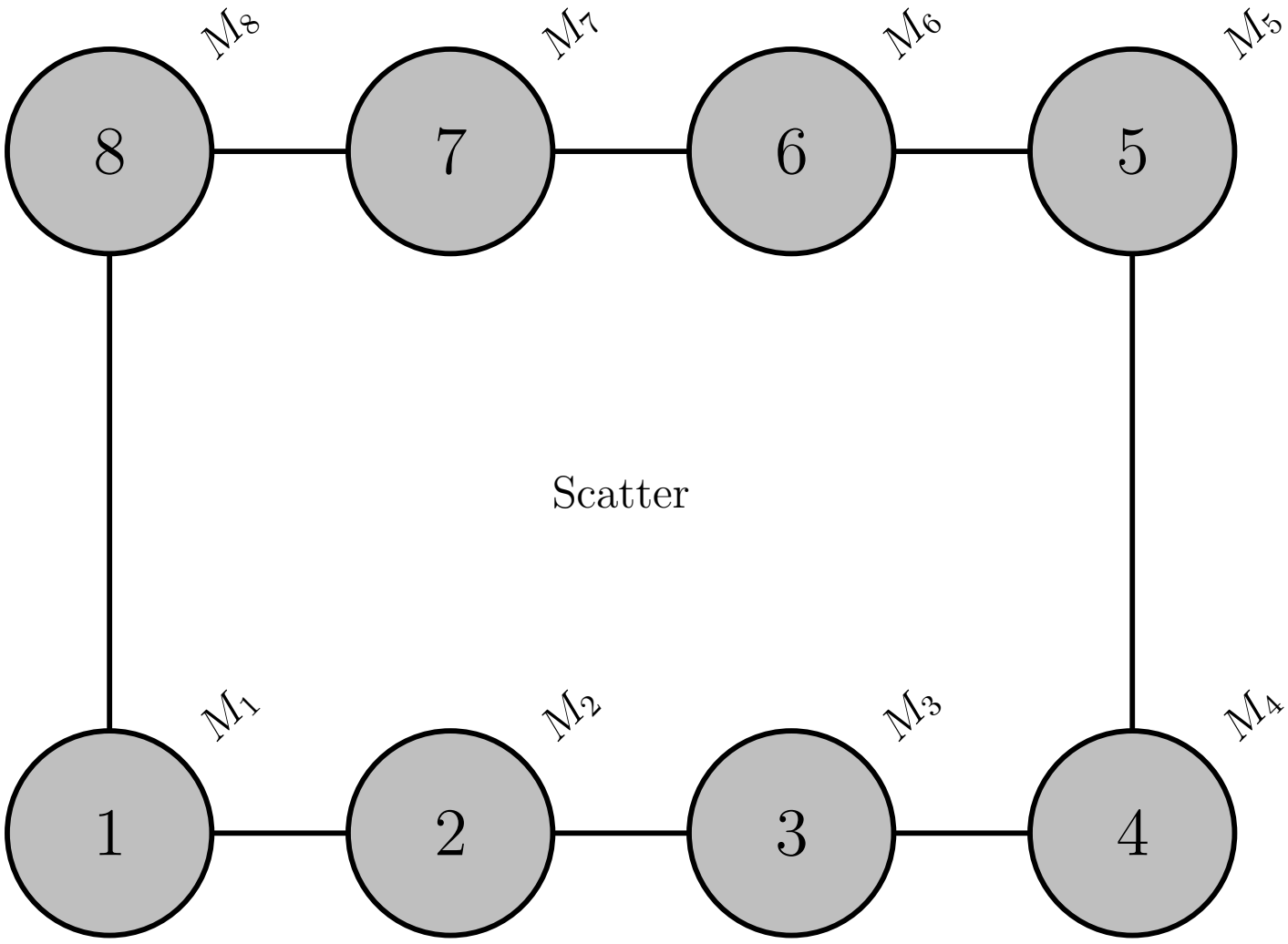
Scatter/gather



Scatter/gather



Scatter/gather

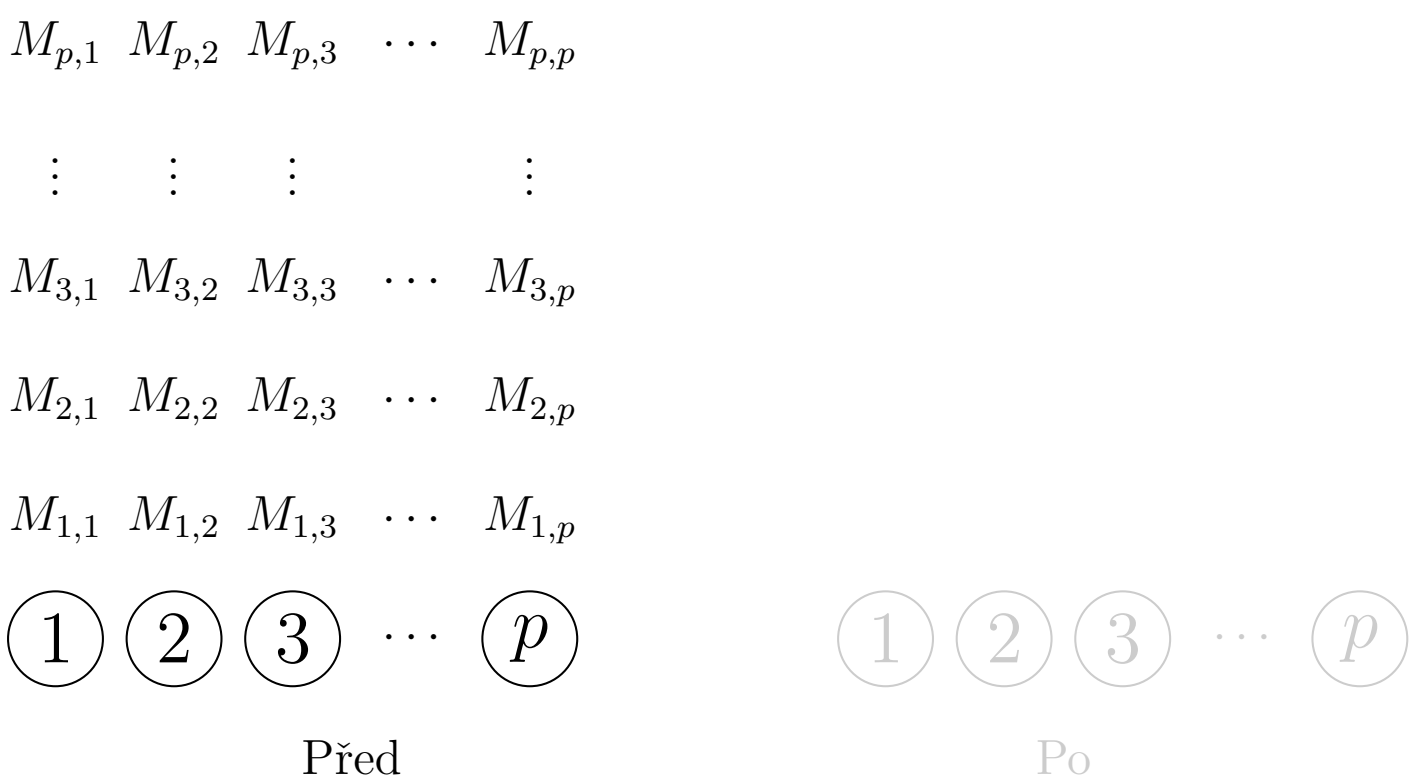


All-to-all personalized communication

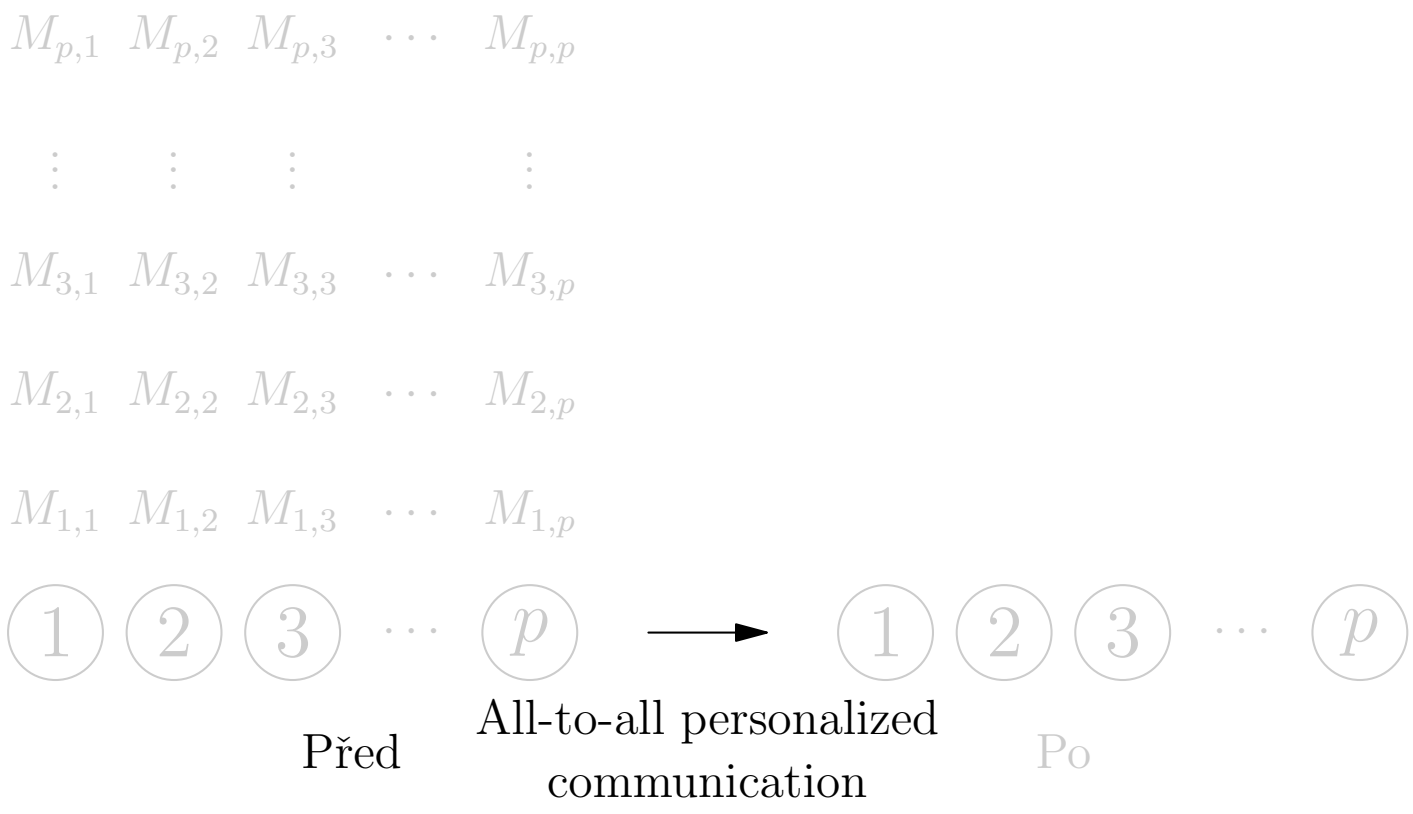
All-to-all personalized communication

- každý proces i má data $M_{i,j}$, která pošle procesu j.

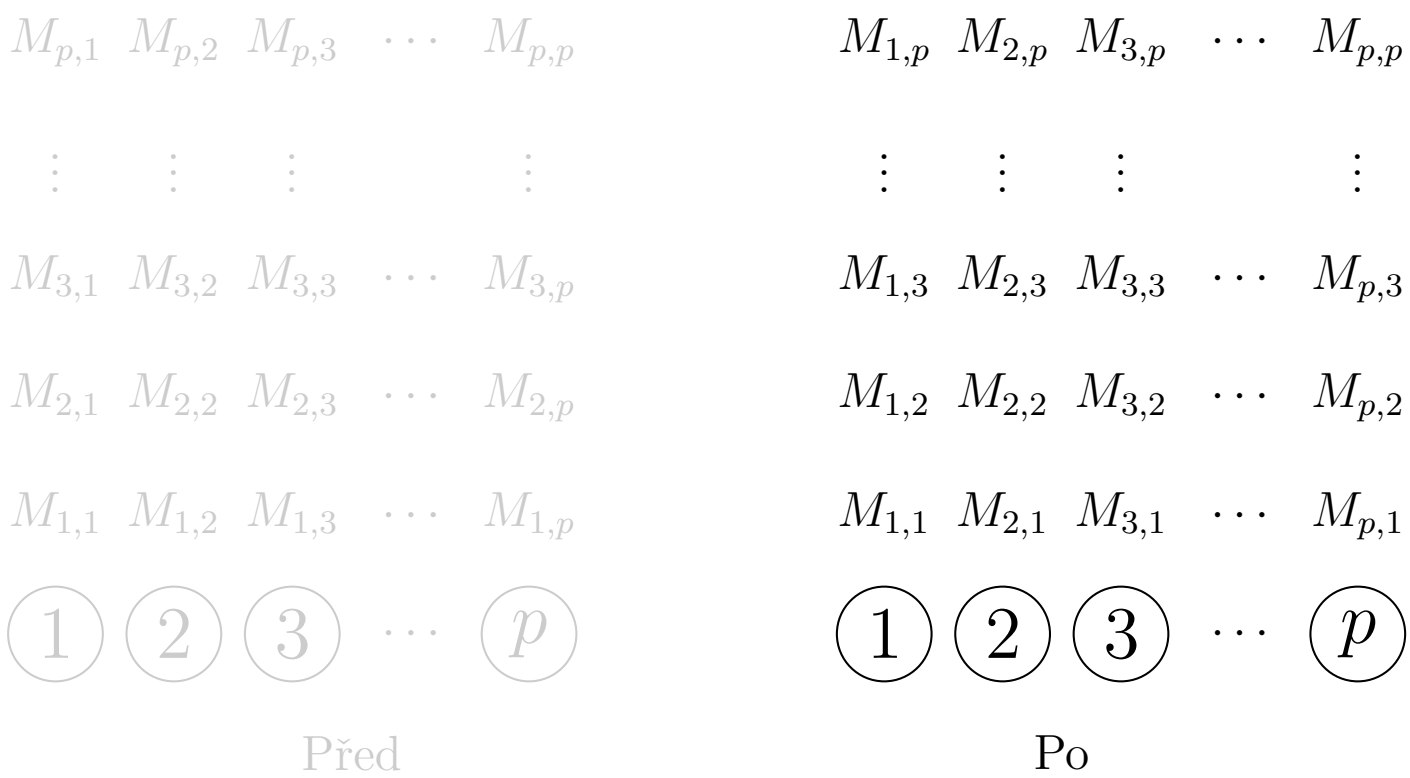
All-to-all personalized communication



All-to-all personalized communication



All-to-all personalized communication



All-to-all personalized communication

- jde vlastně o maticovou transpozici
- stejná analogie jako mezi *scatter* a *one-to-all broadcast* platí i mezi *all-to-all personalized communication* a *all-to-all broadcast*
- z toho plyne i realizace *all-to-all personalized communication* na síti typu **kruh** nebo na ortogonálních sítích

Circular shift

Circular shift

- patří mezi tzv. **permutační komunikační operace**
- každý proces pošle jednu zprávu o velikosti m slov některému jinému uzlu

Příklad: Circular q-shift

- proces i pošle svá data procesu $(i + q) \mod p$