

Multiprocesorski sistemi

“*Directory*” protokoli

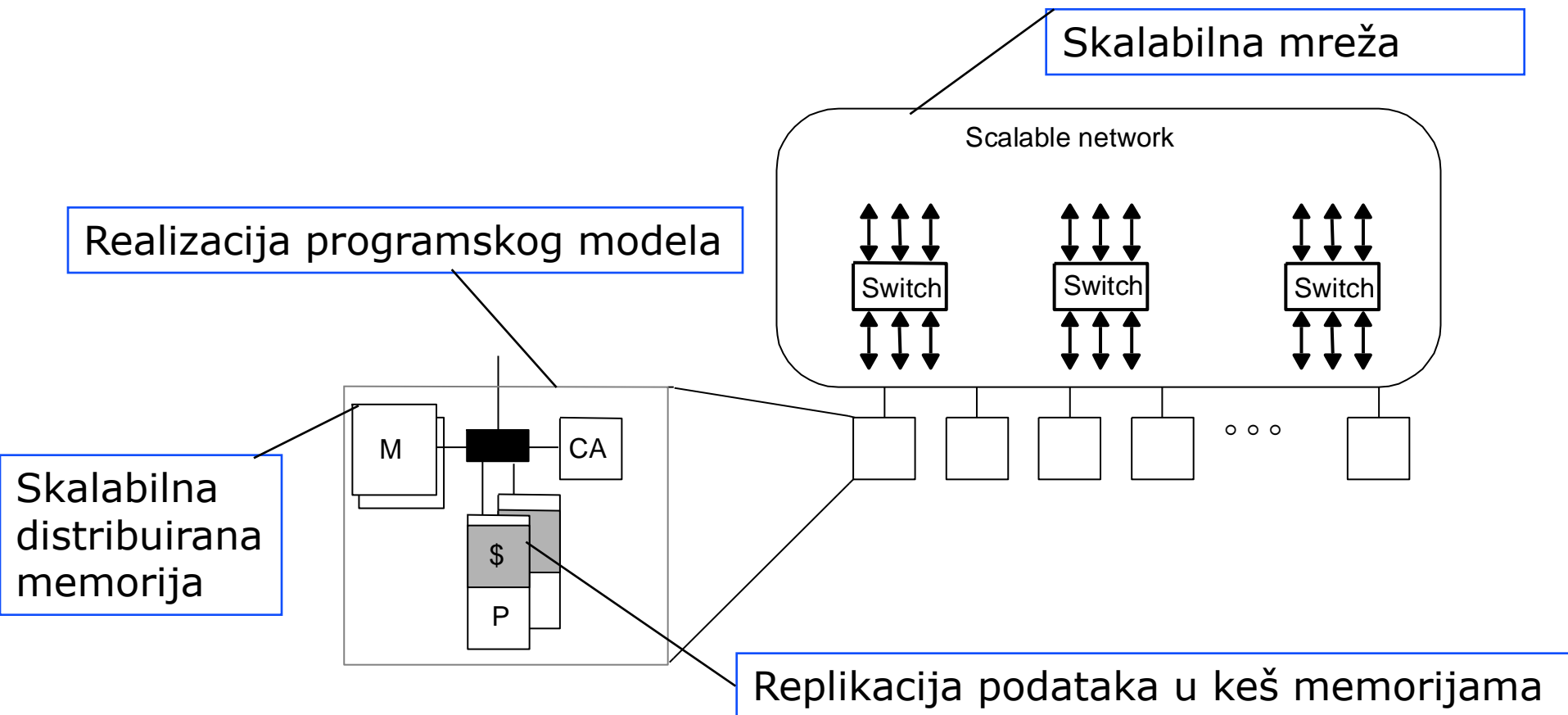
Milo Tomašević

SI4MPS

Skalabilni sistemi

- Sistemi sa zajedničkom magistralom
 - Ograničen propusni opseg
 - Ograničen broj procesora
- Skalabilni sistemi sa više čvorova
 - Memorija distribuirana po čvorovima
 - replikacija podataka u lokalnim keš memorijama
 - Skalabilana interkonekciona mreža
 - omogućava više istovremenih transakcija
 - CA – komunikaciona podrška
 - kroz protokole realizuje programski model interpretacijom transakcija na mreži
- Zahtevi koherentnog sistema?

Skalabilni sistemi



Zahtevi koherentnog sistema

- Skup stanja, dijagram prelaza i akcije
- Akcije:
 - (a) Odluka kada pozvati akcije protokola
 - (b) Pronaći informaciju o stanju drugih kopija podatka
 - Da li je potrebna komunikacija?
 - (c) Locirati druge kopije
 - (d) Komunikacija sa drugim kopijama (inv/upd)
- (a) slično na svim sistemima
 - Na osnovu lokalnog stanja kopije u keš memoriji
 - Akcije protokola se pozivaju ako je ugrožena koherencija
- Različite strategije za (b) - (d)

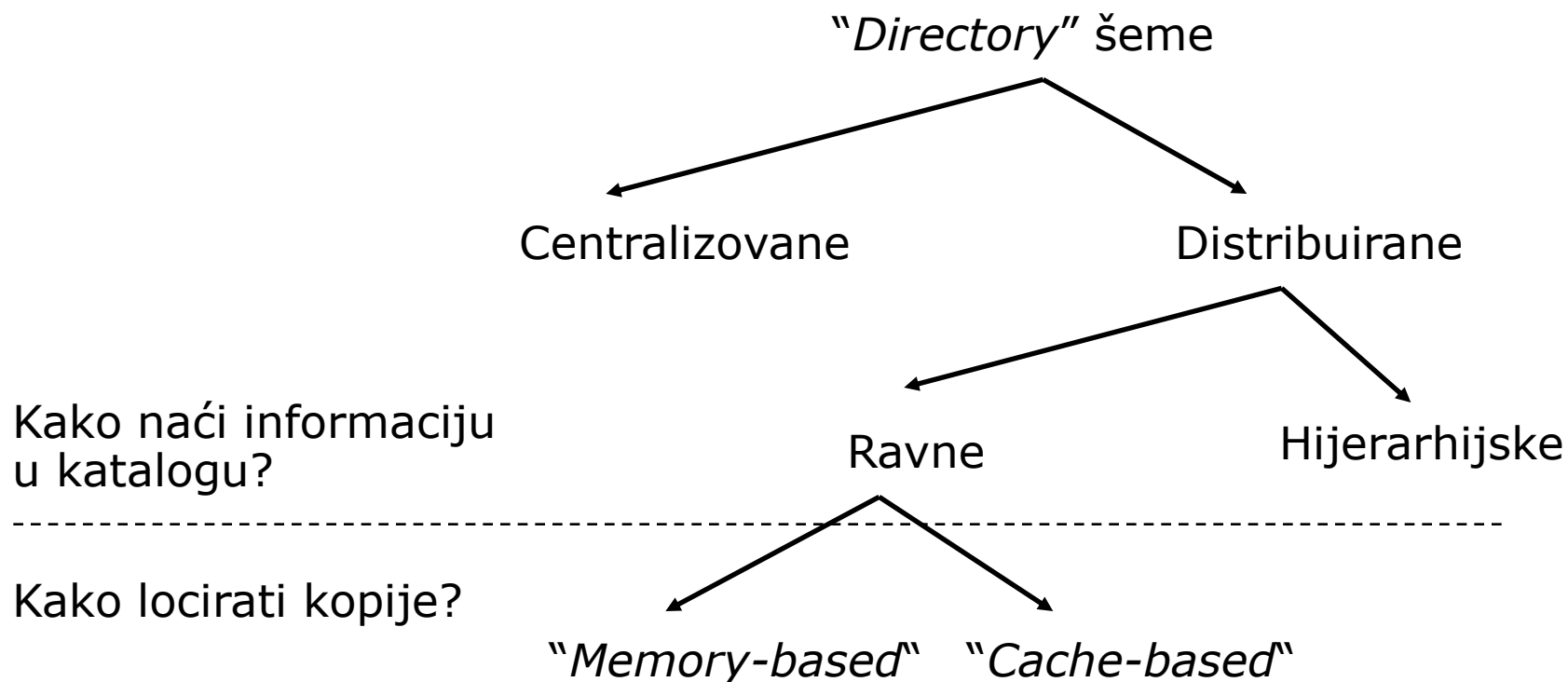
Zahtevi skalabilnog koherentnog sistema

- U sistemu sa zajedničkom magistralom akcije (b) - (d) zasnovane na
 - "broadcast"-u
 - "snooping"-u (najčešće ne rezultira u akciji – *spam!*)
- U skalabilnim sistemima koji nisu zasnovani na zajedničkoj magistrali (CC-NUMA)
 - "Broadcast" je skup (n zasebnih poruka)
 - Nescalabilno rešenje
- Skalabilni protokol
 - Ista ili slična stanja i dijagram prelaza
 - ... ali različita organizacija informacija i implementacija akcija
- Rešenje zasnovano na katalozima (*directory*)

Directory protokoli - principi

- Akcije:
 - (b) Eksplicitna informacija se nalazi u memoriji
 - (c) Lociranje drugih kopija iz ulaza kataloga
 - (d) Komunikacija se ostvaruje "unicast" porukama na poznata odredišta -> skalabilno rešenje (bez spama!)
 - Npr. RM, WH, ...
- Odgovornost za održavanje koherencije:
 - Na centralizovanom kontroleru
 - ... ali i na lokalnim kontrolerima (obično CA)
- Različite projektne odluke
 - Organizacija kataloga
 - Strategija protokola

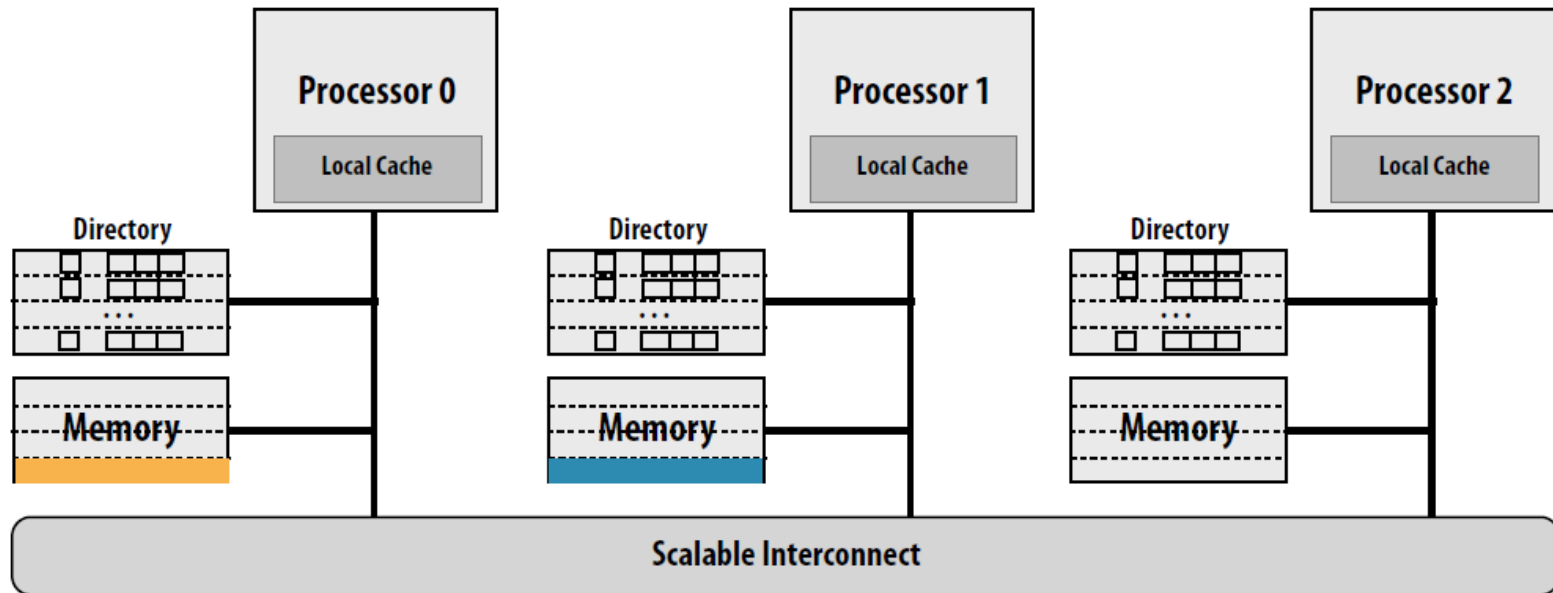
Organizacija kataloga



Organizacija kataloga

- Katalog - centralizovana informacija o stanju bloka
 - Jedan ulaz odgovara jednom keš bloku
 - Sadrži lokacije i stanje keširanih kopija tog bloka
- Centralizovana memorija i katalog
 - Lako pronalaženje, lakša serijalizacija
 - ... ali loša skalabilnost
- Distribuirana memorija i katalog
 - Ravne šeme
 - Hijerarhijske šeme
- Ravne šeme
 - Razlikuju se po performansama i zauzeću memorije
 - Ulazi kataloga samo u memoriji (*memory-based*)
 - Ulazi kataloga u memoriji i keševima (*cache-based*)

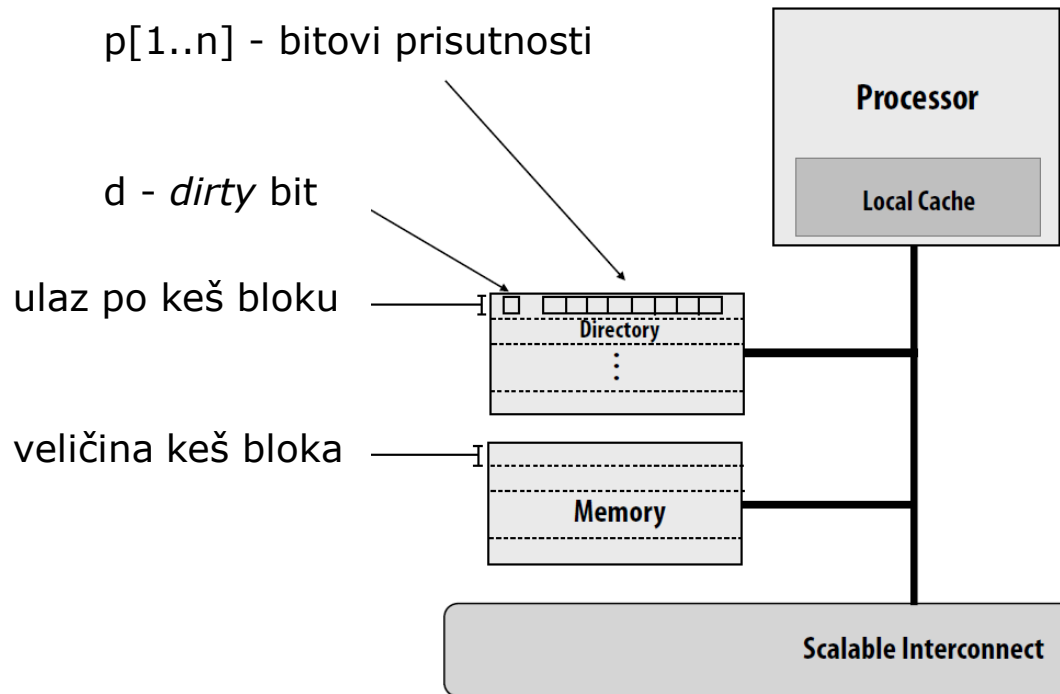
Directory protokoli - principi



○ Ravne šeme

- Katalog uz memoriju, obično u matičnom (*home*) čvoru
- Pristupa mu se na osnovu adrese bloka (npr., heširanjem)
- Poruka se šalje direktno matičnom čvoru (ako je udaljen)

Directory protokoli - organizacija



- Različita organizacija informacije o koherenciji, ali ista stanja
 - Ulaz "*full-map*" kataloga – vektor prisutnosti
 - $n+1$ bit (n bitova prisutnosti + *dirty* bit) - $p[1..n] + d$
 - Dva bita (*valid* + *dirty*) uz blok u keš memoriji

Directory protokoli - operacije

○ Pi: RM B

- Ako je $d = 0$
 - M: send B \rightarrow Pi; $p[i] = 1$
- Ako je $d = 1$ i $p[j] = 1$
 - j (identitet vlasnika) \rightarrow Pi ; Pi:read B \rightarrow Pj ;
 - Pj:send B \rightarrow Pi, M ; Mupd, $p[i] = 1, d = 0$; lokalna stanja $v = 1, d = 0$

○ Pi: WM B

- Ako je $d = 0$
 - M:send B , p \rightarrow Pi; $p[1..n] = 0$ osim $p[i] = 1$; $d = 1$;
 - Pi:inv \rightarrow Pk, $p[k] = 1$; Pk: $v = 0, \text{ack} \rightarrow$ Pi
 - Pi:write B, $v = 1, d = 1$

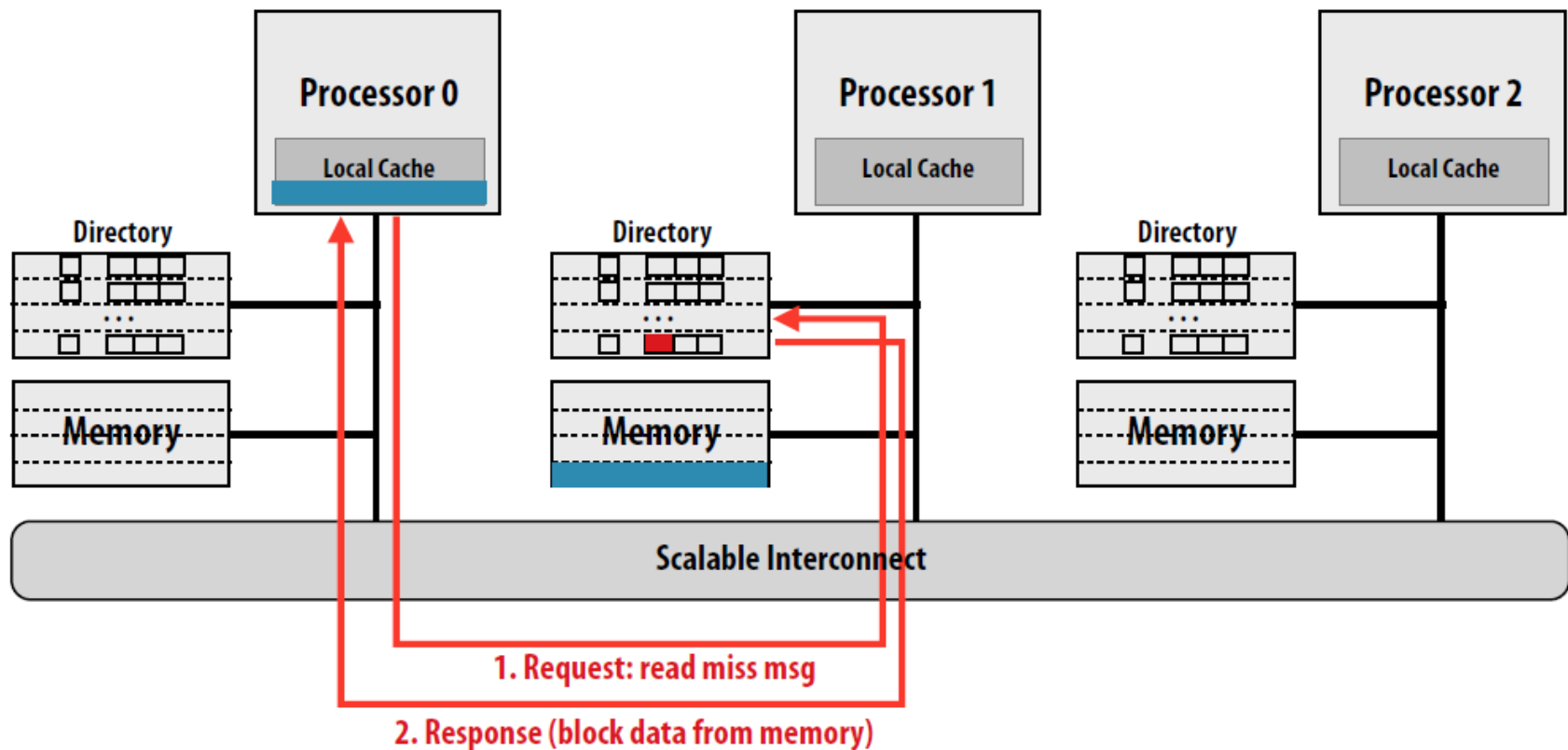
- Ako je $d = 1$ i $p[j] = 1$

- Slično, samo Pi traži i dobija blok od Pj, koji ga zatim invaliduje

○ Pi: WH B – slično WM, ali se ne šalje blok (*upgrade*)

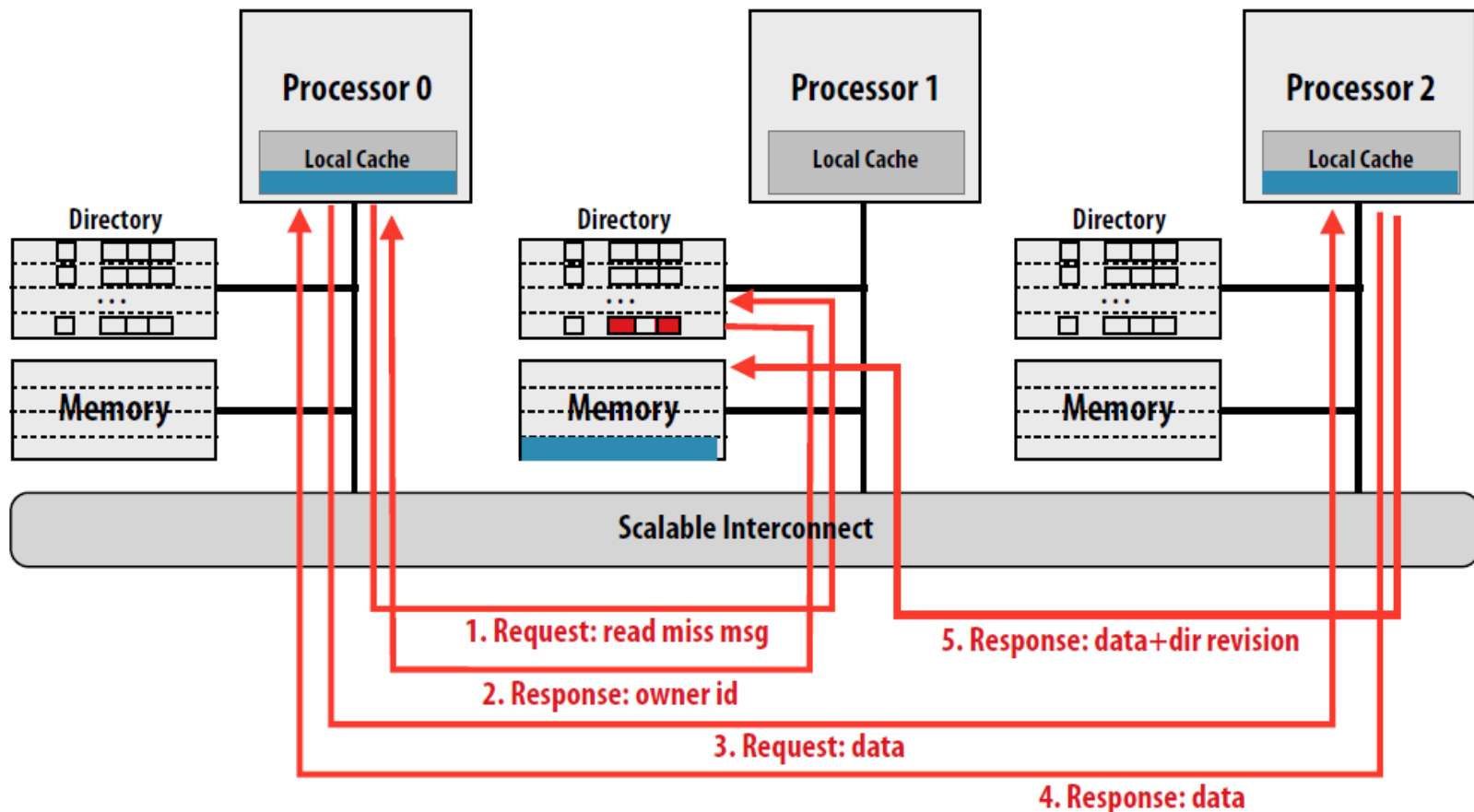
Directory protokoli – RM (1)

Promašaj pri čitanju, kopija u memoriji ažurna ($d = 0$)



Directory protokoli – RM (2)

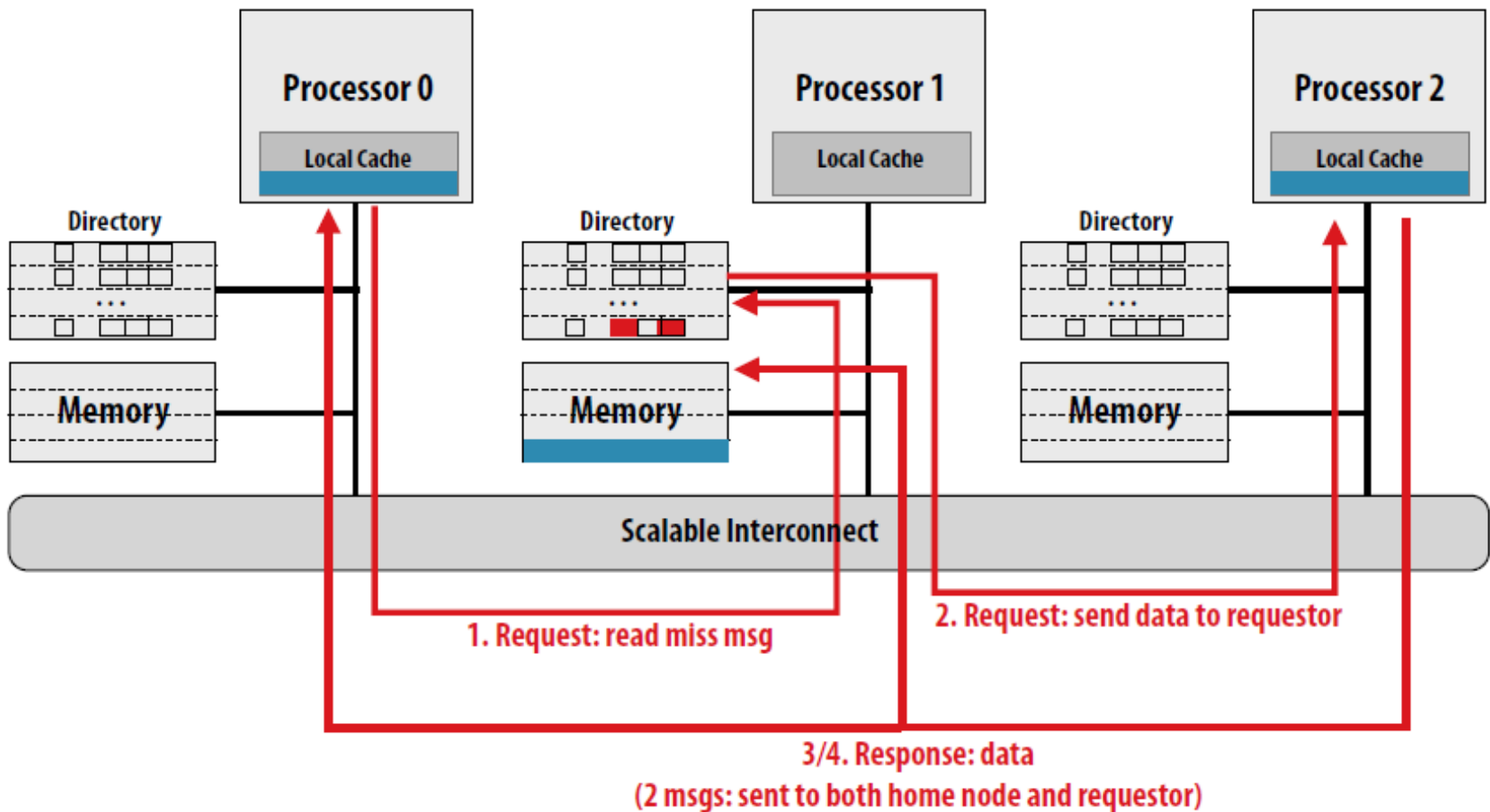
Promašaj pri čitanju, kopija u memoriji neažurna ($d = 1$)



Directory protokoli – RM (3)

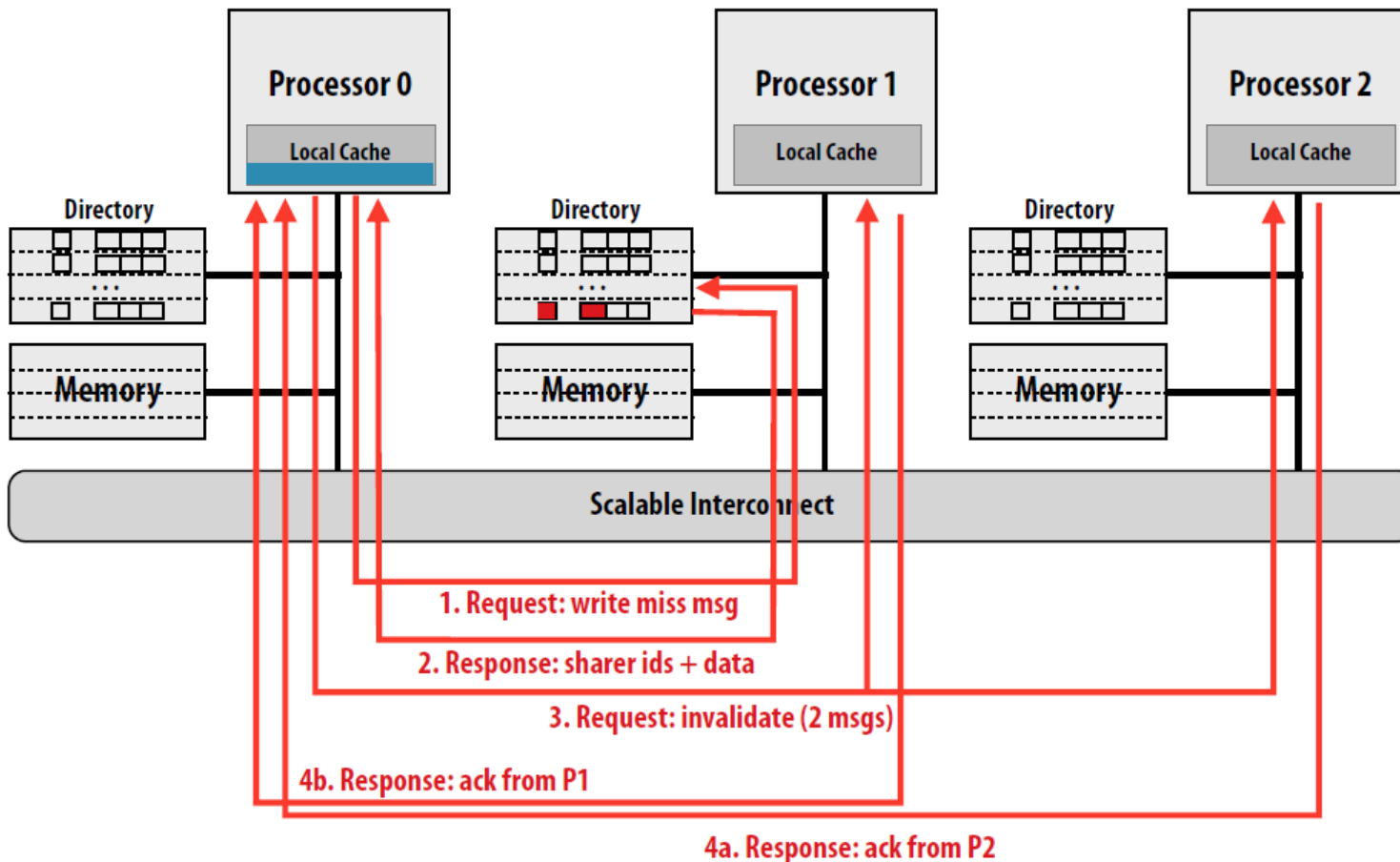
- Performanse ove operacije
 - Broj poruka
 - Latencija – broj poruka na kritičnom putu (sekvenca zavisnih operacija da bi se opslužio zahtev)
- Od 5 poruka, 4 su na kritičnom putu
 - Poruke 4 i 5 mogu u paraleli
- Optimizacija
 - Prosleđivanje zahteva (*request forwarding*)
 - Smanjuje broj poruka i latenciju
 - Od 4 poruka, 3 su na kritičnom putu (Poruke 3 i 4 mogu u paraleli)

Directory protokoli – RM (4)



Directory protocols – WM

Promašaj pri upisu, kopije u memoriji ($d = 0$), P1 i P2 ažurne



Directory protokoli - operacije

- Zamena kod Pi
 - Ako je lokalni $d = 1$
 - Ažuriranje memorije, $p[i] = 0$; $d = 0$
 - Ako je lokalni $d = 0$
 - Može se (ali ne mora) obavestiti memorija (konzervativno!)
 - + Nema nepotrebnih *inv* poruka, oslobađa pointere u katalogu
 - Nepotrebno za podatke koji se samo čitaju, dodatni saobraćaj
- Implementacija složena
 - Serijalizacija mnogo složenija
 - Dosta prelaznih stanja (desetine ... , česti prelazi)
 - Mogući problemi "trke" kada informacija u katalogu nije sasvim ažurna (npr., pri zameni)
 - Vlasnik izbacuje blok, drugi traži dozvolu za upis
 - Rešava se odbijanjem (NACK)

Directory protokoli - skaliranje

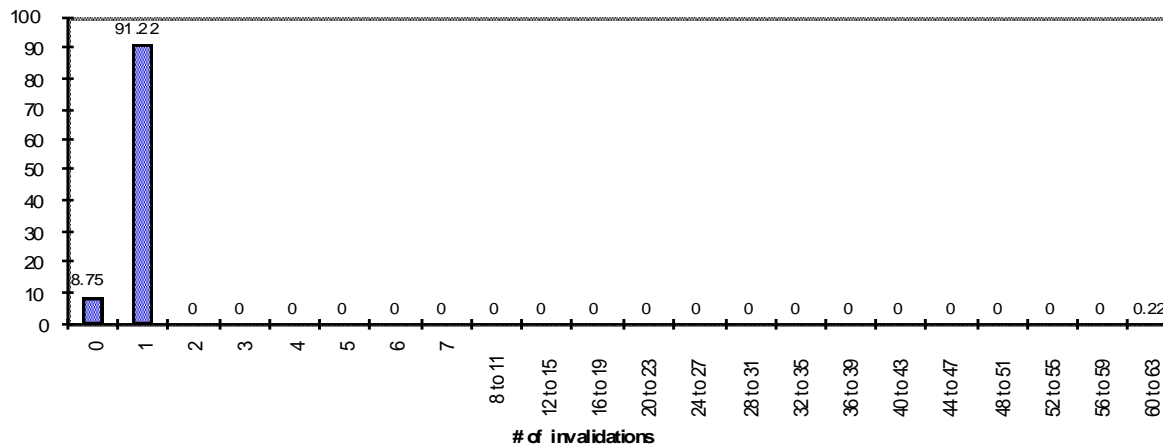
- Skaliranje propusnog opsega memorije i kataloga
 - Centralizovani katalog – usko grlo!
 - Kako distribuirati informaciju iz kataloga?
- Skaliranje karakteristika performanse
 - *Saobraćaj*: broj transakcija u mreži koje generiše protokol
 - *Latencija*: broj transakcija koje su na kritičnom putu
- Skaliranje prostora potrebnog za katalog
 - Broj bitova raste linearno sa brojem procesora
 - ... ali i veličina memorije
- Organizacija kataloga
suštinska za skalabilnost protokola

Directory protokoli - skaliranje

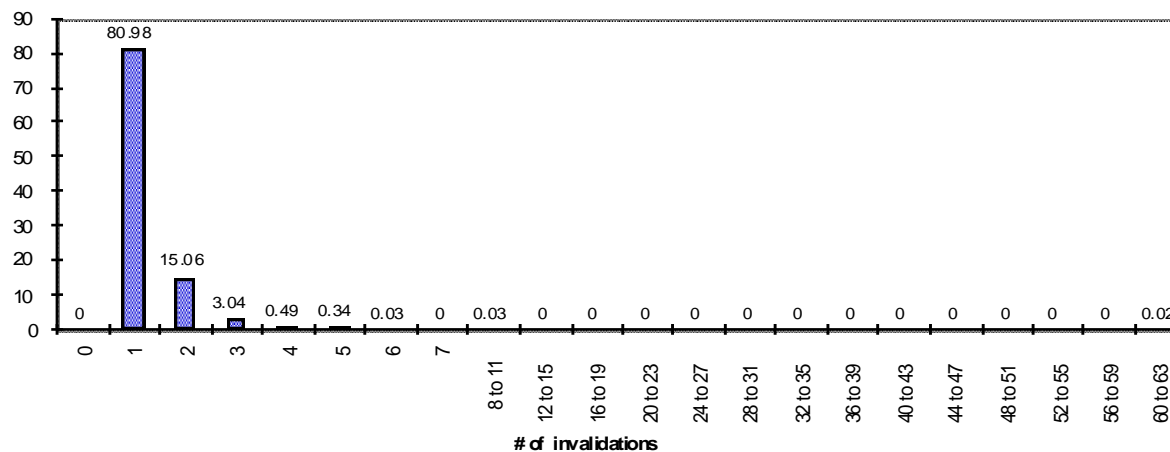
- Koliko se dobija izbegavanjem "broadcast"-a?
- Bitne inherentne karakteristike aplikacija
 - Frekvencija operacija kada su potrebne invalidacije
 - Prosečan broj invalidacija po takvoj operaciji
 - Kako ova dva indikatora skaliraju?
- Ukazuju na to kako organizovati katalog
- Simulacija sa aplikacijama iz SPLASH-2
 - Keš memorije beskonačne veličine
 - Sistemi sa 64 procesora
- Važan zaključak
 - Broj aktivno deljenih kopija obično dosta mali!
 - Sporo raste sa povećanjem broja procesora

Načini deljenja u aplikaciji

LU Invalidation Patterns

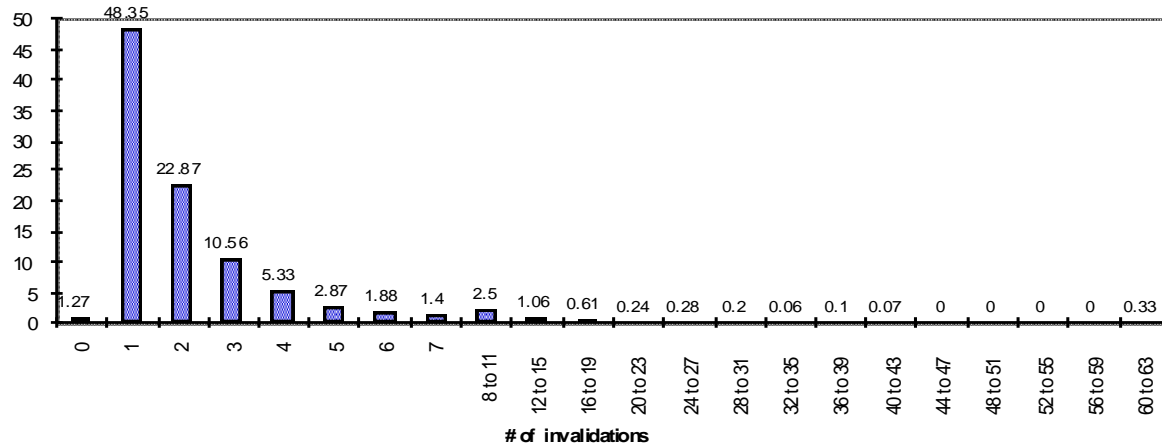


Ocean Invalidation Patterns

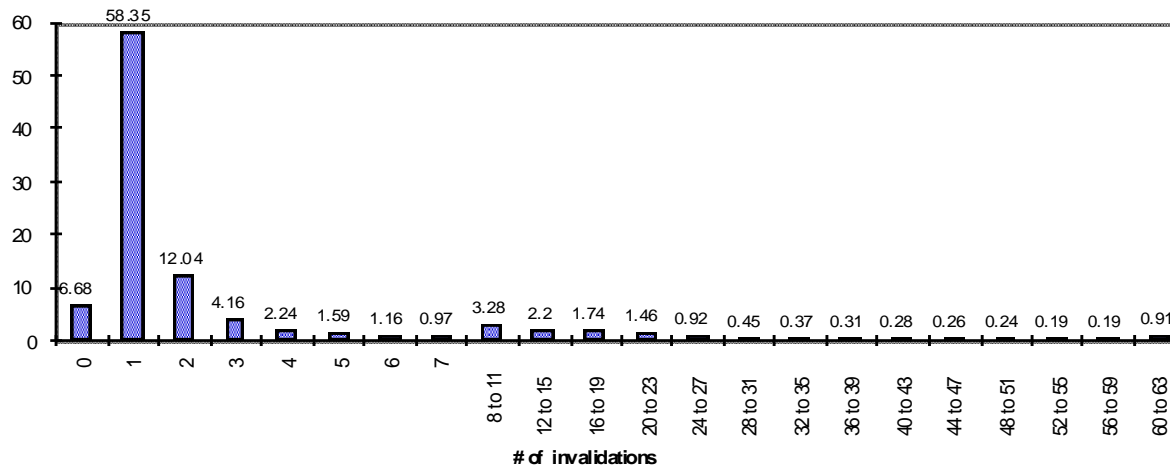


Načini deljenja u aplikaciji

Barnes-Hut Invalidation Patterns



Radiosity Invalidation Patterns

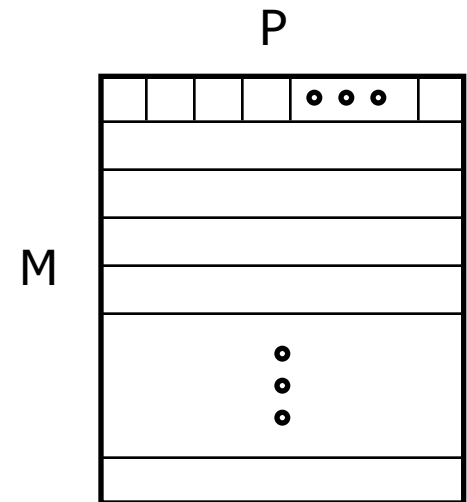


Načini deljenja u aplikaciji

- Načini deljenja sa aspekta broja invalidacija
 - Kod i *"read-only"* objekti – nema invalidacija
 - Migratorni objekti – 1-2 invalidacije po upisu
 - *"Mostly-read"* objekti – dosta invalidacija, ali retko
 - Često upisivani i čitani objekti – invalidacije česte, ali malo deljenih kopija može da se napravi između njih
 - Sinhronizacioni objekti – ako je velika kontencija za njih, neophodna posebna SW ili HW podrška
- U multiprogramskim serverima problem još lakši
 - Migracija procesa nameće samo jednu invalidaciju
- Zaključci analize
 - Skaliranje saobraćaja i latencije nije loše
 - Moguće smanjiti veličinu kataloga

Ravne “*memory-based*” šeme - skaliranje

- Informacija o keširanim kopijama u matičnom čvoru
 - Slično centralizovanim šemama, ali distribuirana
- Skaliranje performanse
 - Saobraćaj pri upisu: srazmeran broju deljenih kopija
 - Latencija upisa: invalidacije mogu da se šalju u paraleli
 - Performanse najbolje
- Skaliranje prostora u “*full-map*” šemi
 - Prostorni “*overhead*” za blok od 64 bajta:
 - $P = 64 \rightarrow 12.5\%$
 - $P = 256 \rightarrow 50\%$
 - $P = 1024 \rightarrow 200\%$
 - Prostorni “*overhead*” – $O(P \cdot M)$
 - Loša skalabilnost



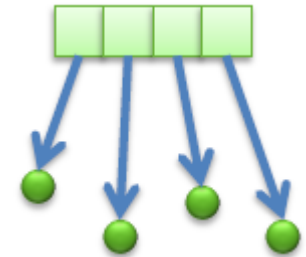
Smanjivanje prostora za katalog

- Kako smanjiti "visinu" M i "širinu" P ?
- Optimizacije za osnovnu "full-map" tehniku
 - Povećanje veličine bloka smanjuje broj ulaza, ali povećava lažno deljenje
 - Multiprocesorski čvorovi (1 bit/MP čvor, ne 1 bit/PE!)
 - Opet skalira kao $P \cdot M$, ali ipak prihvatljivo za velike mašine
 - $P = 256$, 4 PE čvor, 128 B blok -> 6.25% overhead
- Smanjivanje "visine" kataloga M
 - Broj memorijskih blokova >> broj keširanih blokova
 - Mnogi ulazi kataloga se ne koriste u datom vremenu
 - Umesto rezervisanja ulaza za svaki memorijski blok, organizovati katalog na principu keš memorije

Smanjivanje prostora za katalog

○ Smanjivanje "širine" kataloga P

- Za većinu blokova samo mali broj keširanih kopija (često do 5)
- Umesto posebnog bita za svaki PE, ulaz kataloga sadrži manji broj pokazivača (svaki $\log P$ bita) na deljene kopije
- Npr., $P=1024 \Rightarrow$ 10-bitni pointeri, čak i 100 pokazivača malo štedi prostor
- Analize načina deljenosti ukazuju da je nekoliko pokazivača dovoljno (često do 5)
- Neophodna strategija prekoračenja kada ima više deljenih kopija nego pokazivača



○ Dir_i šeme sa ograničenim brojem pokazivača

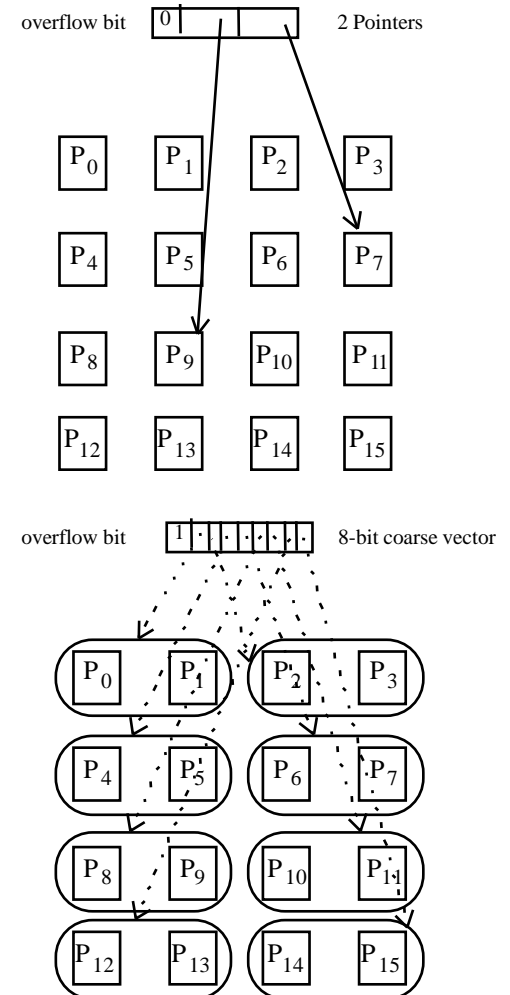
- Ograničen broj pointera i po ulazu (obično $i \ll P$)
- X - strategija prekoračenja (*pointer overflow*)
- Skalabilne po veličini kataloga – $O(M \cdot \log P)$
- Degradacija performansi kod intenzivne deljenosti

Smanjivanje širine kataloga

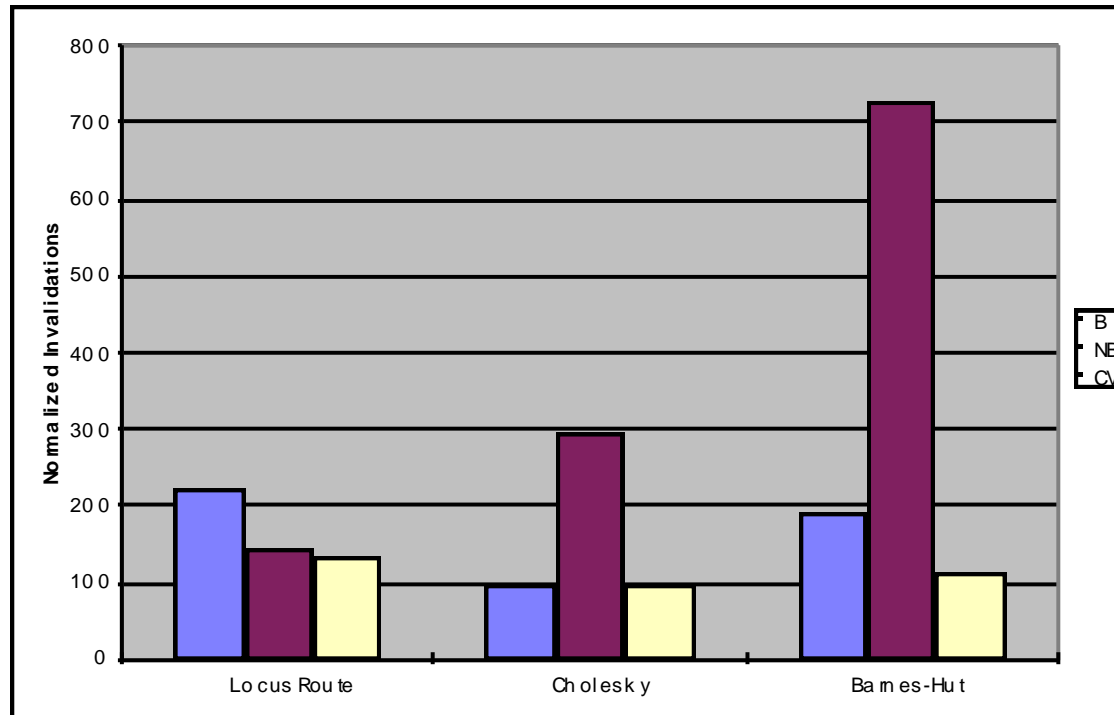
- *Dir_i NB* (Agarwal et al.)
 - Pri prekoračenju pointera, oslobađa jedan pointer invalidacijom jedne deljene kopije
 - Izbegava "broadcast"
 - Ograničava broj deljenih kopija, čak i za čitanje
 - Degradacija performansi kod intenzivne deljenosti
- *Dir_i B* (Agarwal et al.)
 - "Broadcast" bit (b) se postavlja pri prekoračenju pointera
 - Kada je b=1, naredni upis rezultuje "broadcast"-om invalidacija svim procesorima (mnoge nepotrebne!)
 - Ne ograničava broj deljenih kopija za čitanje
 - Jednostavna šema, ali ...
 - povećava saobraćaj u mreži
 - povećava latenciju upisa

Smanjivanje širine kataloga

- $Dir_i CV_r$ (Gupta et al.)
 - Šema sa "grubim" (*coarse*) vektorom
 - Format ulaza zavisi od broja kopija
 - i pointera do prekoračenja, a onda
 - ... svaki bit označava grupu od r PE
 - $r = p/(i \cdot \log p)$
 - Bit se setuje ako bilo koji PE ima kopiju
 - Invalidacije ograničene na grupu PE (nema "broadcast"-a)
 - Neke mogu biti nepotrebne
 - Pogodne za ...
 - Multiprogramska okruženja
 - Aplikacije sa dobrom lokalnošću podataka



Smanjivanje širine kataloga - evaluacija



- 64 PE, 4 pointera
- Broj invalidacija normalizovan prema *full-map* šemi

Smanjivanje širine kataloga

- *Dir_i SW*
 - Pri prekoračenju pointera, SW prekid
 - Pamte se pointeri u lokalnoj memoriji i setuje OW bit
 - Pri upisu, ako je OW = 1 koriste se i zapamćeni pointeri
 - Nema nepotrebnog saobraćaja
 - Cena prekida i SW obrade
 - Povećava zauzetost PE, a time i latenciju i kontenciju
 - Npr., LimitLESS (MIT Alewife)
 - 5 pointera i lokalni bit
 - SW emulacija u heš tabeli
 - Podrška za višenitno izvršavanje
 - Latencija za čitanje iz udaljenog čvora 40-425 ciklusa
 - 84 ciklusa za 5 invalidacija (HW)
 - 707 ciklusa za 6 invalidacija (HW+SW)

Smanjivanje širine kataloga

○ *Dir_i DP*

- Vrlo mali broj fiksnih pointera (može i bez njih!)
- Lista slobodnih dinamičkih pointera u memoriji (*pointer/link store*)
- Pri prekoračenju se dodaje kružna ulančana lista dinamičkih pointera
- HW mehanizam + protokol procesor
- Ako ponestane dinamičkih pointera, jedan se oslobađa
- Problem pointera na kopije koje su zamenjene
- Skalabilnost kataloga (P/L prostor srazmeran broju keširanih kopija)
- Npr., Stanford FLASH

Smanjivanje visine kataloga

- Broj keširanih blokova relativno mali u odnosu na ukupan broj
 - Vrlo mala iskorišćenost ulaza kataloga
- Retki (*sparse*) katalog
 - Katalog kao keš memorija
 - Alokacija ulaza dinamički samo kada ima keširane kopije
 - Manji broj ulaza, može i SRAM
- Specifičnosti u odnosu na keš podataka
 - Nema *write-back* (invalidacije pri zameni)
 - Jedan ulaz po bloku (nema prostorne lokalnosti)
 - Obraduje zahteva od svih procesora
 - Potrebna veća asocijativnost i dovoljna veličina da se smanje verovatnoća kolizije

Smanjivanje visine kataloga

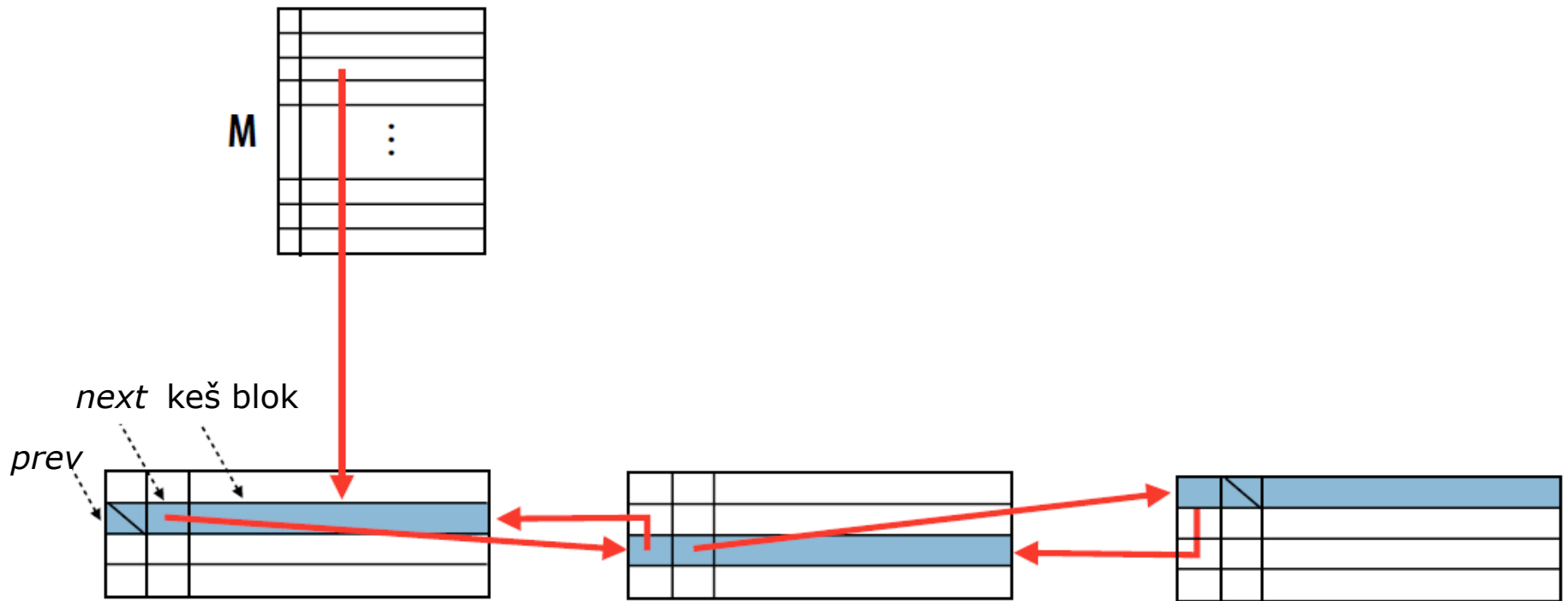
- Performanse
 - Smanjenje veličine kataloga za bar red veličine
 - Kolizije u katalogu donose 15-20% dodatnog saobraćaja
- Tehnike za smanjivanje širine i visine su ortogonalne
- Kombinovani katalog
 - Dve asocijativne keš memorije:
 - Veći keš sa ulazima sa ograničenim brojem pointera (D1)
 - Manji keš sa ulazima u vidu vektora prisutnosti (D2)
 - Prvo se alokira ulaz u D1
 - Pri prekoračenju pointera alokira se ulaz u D2, a oslobađa ulaz u D1
 - Performanse mogu biti bliske "*full-map*" šemi

Ravne “*cache-based*” šeme

- Ulaz kataloga za jedan blok distribuiran
 - ... u vidu ulančane liste
 - U matičnom čvoru samo zaglavlje liste (prva kopija)
 - Svaka kopija u kešu pokazuje na sledeću
- Čitanje
 - Blok se dostavlja iz memorije ili sa čela liste (ako je $d=1$)
 - Nova deljena kopija ide na početak liste
- Upis
 - Propagacija invalidacija (ili ažuriranja) kroz listu
 - Upisana kopija ide na početak liste
- Zamena
 - Izlančavanje iz liste (lakše u dvostruko ulančanoj listi)

Ravne “*cache-based*” šeme - primer

- Scalable Coherent Interface (SCI) IEEE Standard
 - Dvostruko ulančane liste (pokazivači *prev* i *next*)



Ravne “*cache-based*” šeme - skaliranje

○ Prednosti

- Manji prostor za katalog (manje pokazivača)
 - Broj mem. blokova * 1 + ukupan broj keš blokova * 2
- Poredak pristupa poznat
- “*Overhead*” invalidacija decentralizovan
- IEEE standard

○ Problemi

- Latencija pri upisu srazmerna broju deljenih kopija
- Povećana zauzetost
(čak i promašaj pri čitanju angažuje prvi čvor)
- Potrebna sinhronizacija pri zameni susednih kopija
- Kompleksne implementacije (npr., NUMA-Q, Exemplar)