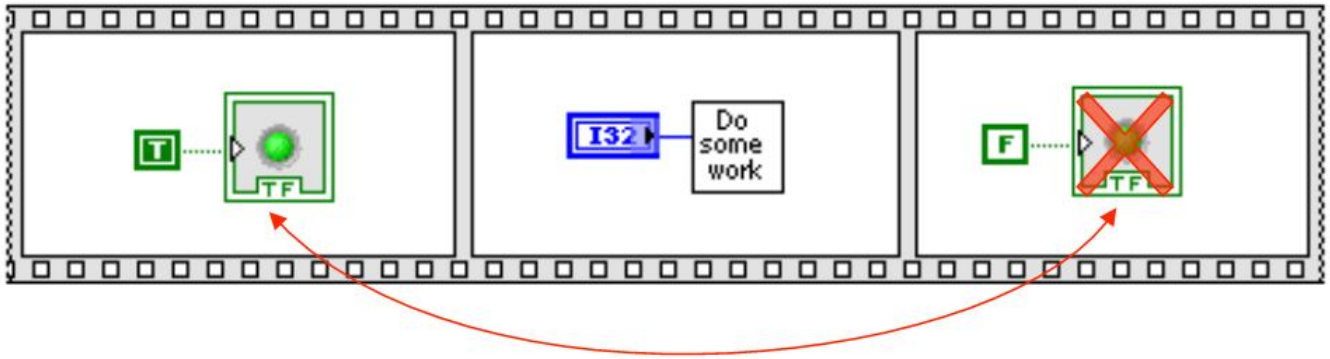
# PROGRAMSKI PAKET LabVIEW

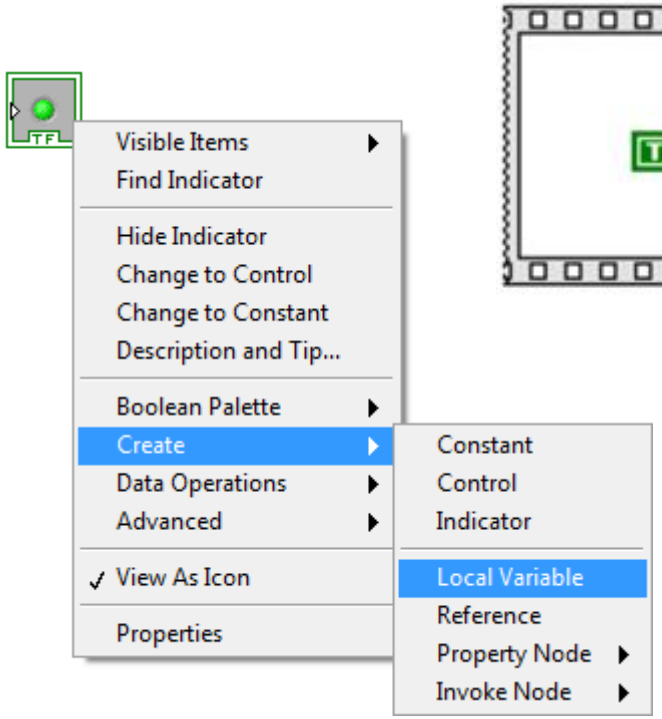
## Podsetnik

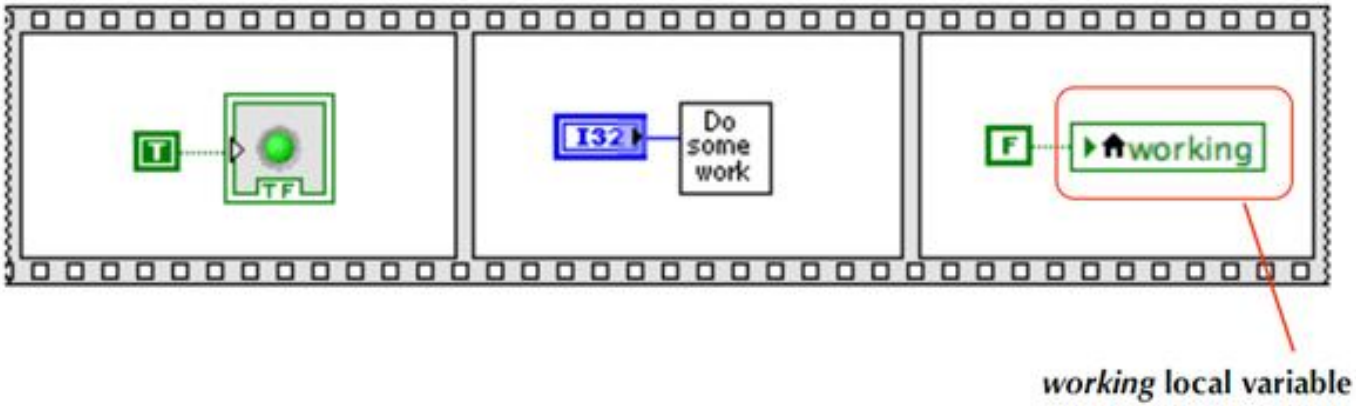
## Variables, Synchronization, Design Patterns, Property and Invoke Nodes, Control Reference

# Lokalne promenljive




Nije moguće sa istim indikatorom.



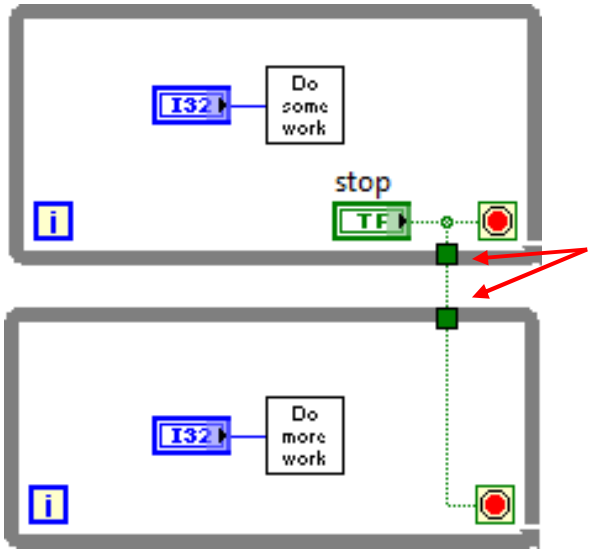




*working* local variable

- Lokalna promenljiva je povezana sa odgovarajućom kontrolom/indikatorom.
- U lokalnu promenljivu se može upisati vrednosti i iz nje se može iščitati vrednost bez obzira sa tipom terminala sa kojim je povezana

# Lokalne promenljive

- Omogućava zaustavljanje dve petlje istovremeno.

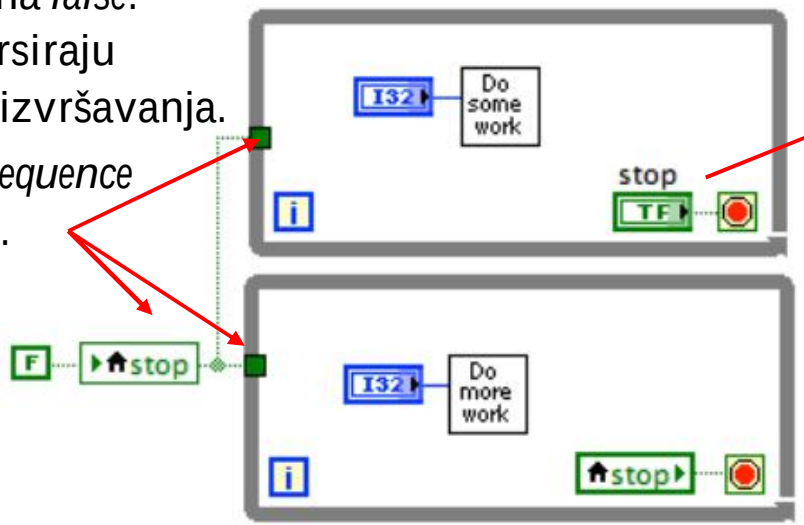


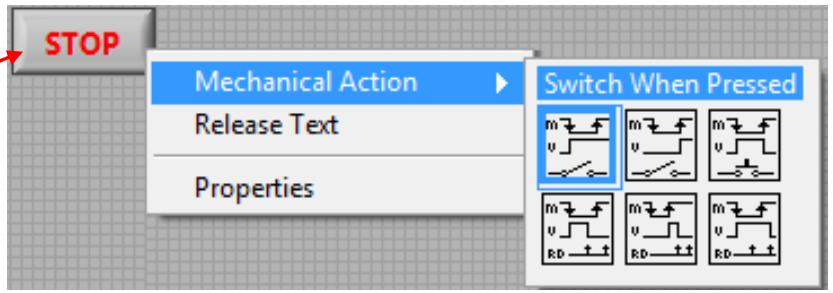

Tunel postaje aktivan tek kada se završi gornja petlja. Tek tada počinje da se izvršava donja petlja i to samo jednom.

Oznaka za lokalnu promenljivu.

Inicijalizacija Stop kontrole na *false*. Tuneli forsiraju redosled izvršavanja.

Umesto *sequence* strukture.

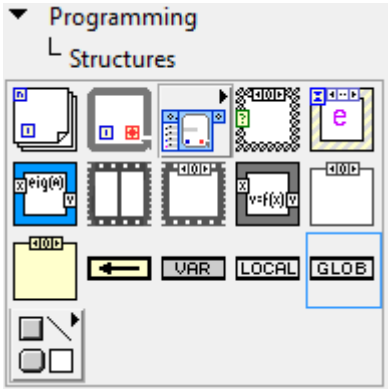



Mora biti *Switch*, jer *Latch* zadržava vrednost dok LabVIEW ne iščita vrednost i resetuje je.

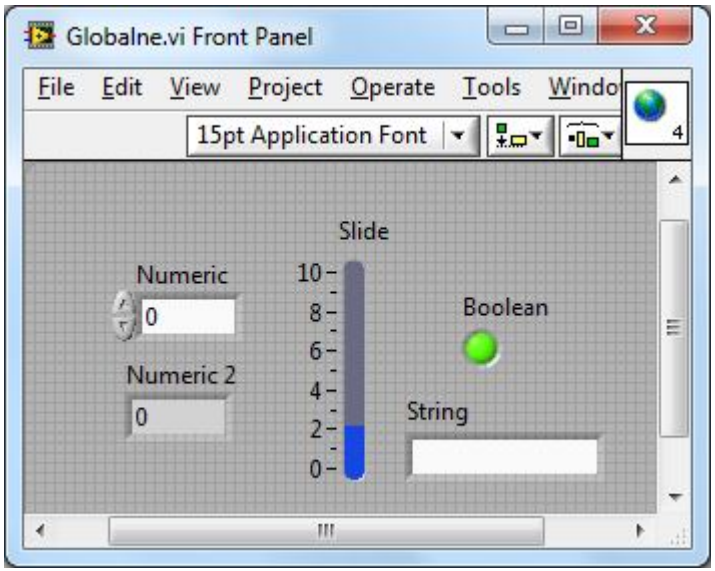
# Globalne promenljive

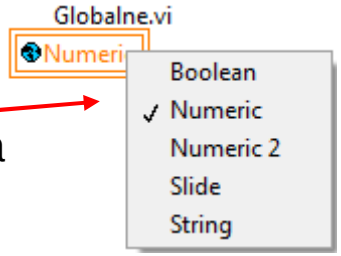
- Lokalna promenljiva je vezana za odgovarajući terminal i važi samo u VI koji sadrži taj terminal (kontrolu / indikator).
- Globalna promenljiva ima opseg važenja na nivou LabVIEW aplikacije. Globalna premenljiva iz jednog projekta se može pozvati u VI drugog projekta.
- Globalne promenljive omogućuvaju razmenu podataka između različitih VI koji se izvršavaju u toku jedne aplikacije.




Fajl Globalne.vi može sadržati više terminala različitog tipa, odnosno predstavlja skup promenljivih.

Nema BP.

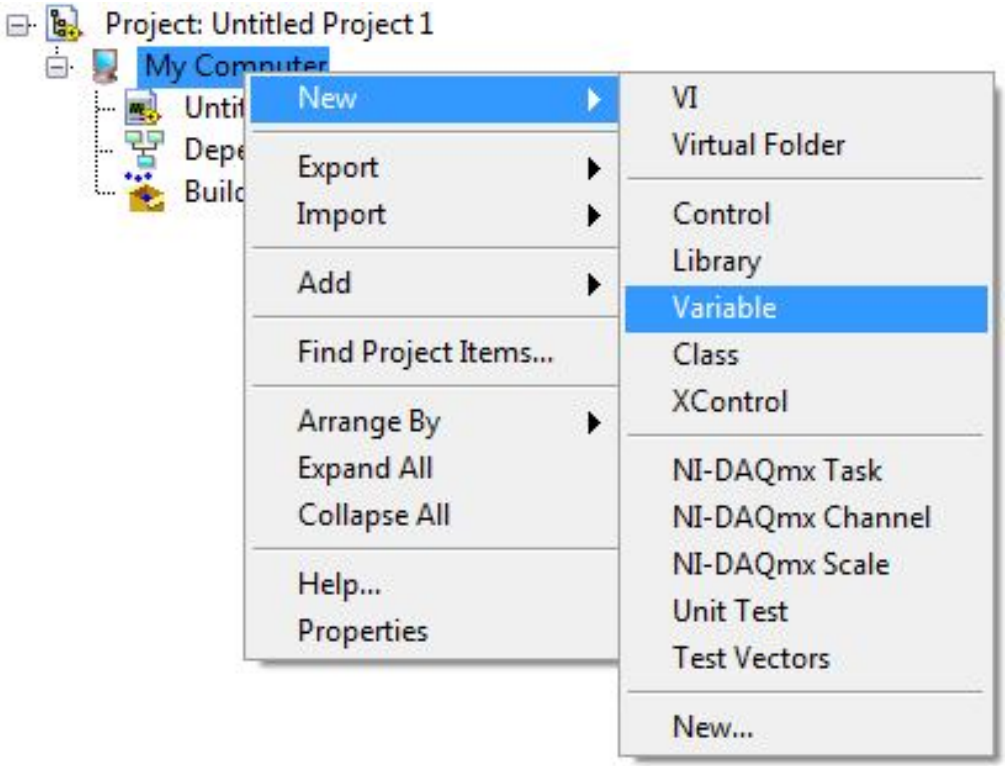




Izbor promenljive iz skupa promenljivih.

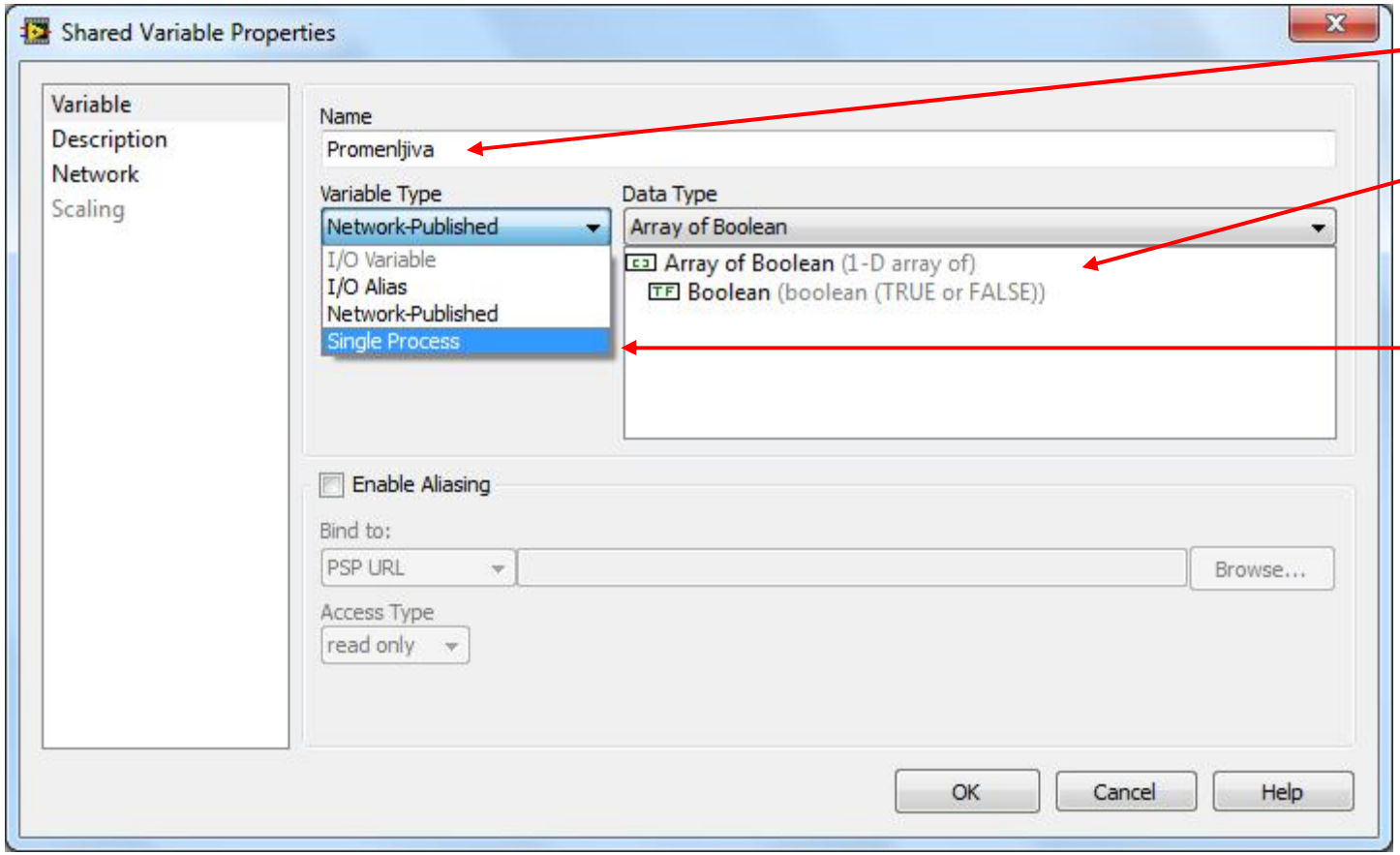
Oznaka za globalnu promenljivu.

## Shared variables

- Za razliku od globanlih promenljivih *shared variables* se mogu koristi na više umreženih računara čime se omogućava mrežno deljenje resursa.
- Vezana je za projekat



# Shared variables

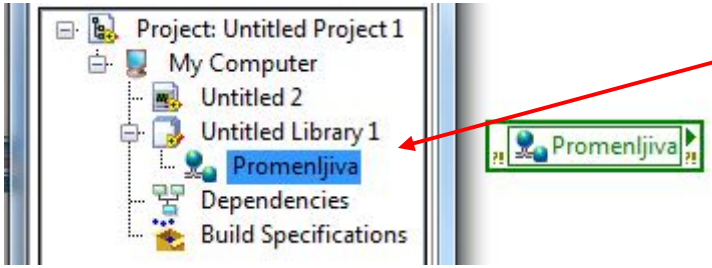



Naziv promenljive

Tip podatka promenljive

*Single process* – isto što i globalna promenljiva, jedino što vrlo lako postaje vidljiva na mreži, bez izmena koda.

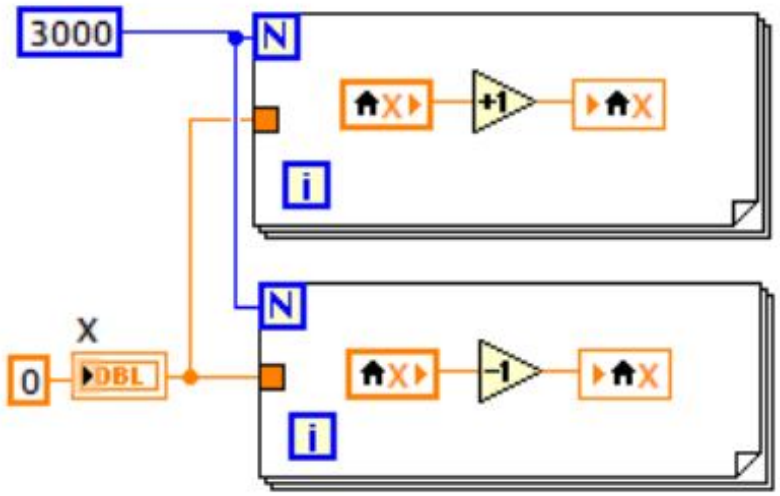
*Network-Published* – vidljiva na mreži.




Prevlačenje na *BP*

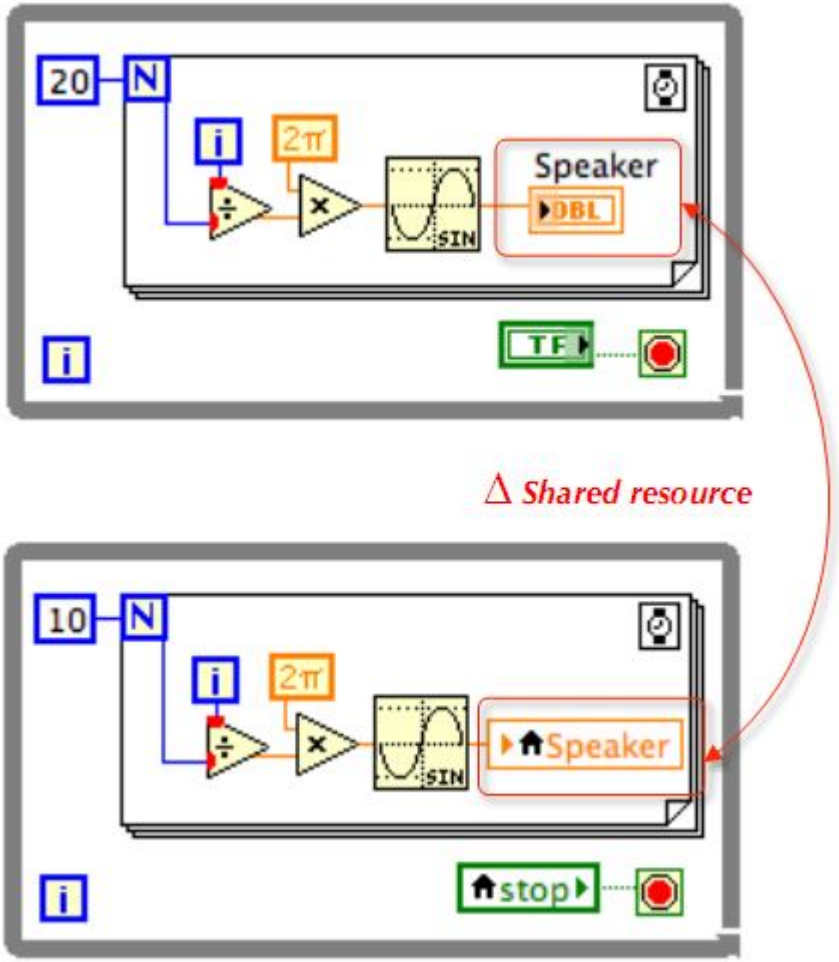
## Race condition

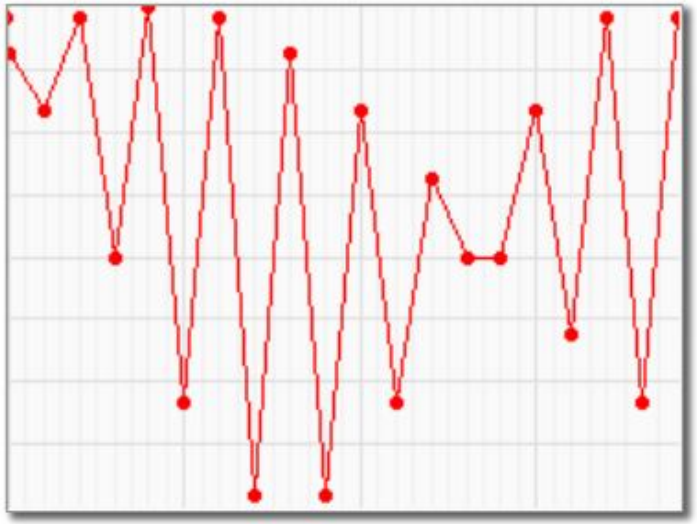
- Paralelno programiranje omogućava brže izvrašavanje nekog zadatka, međutim mogu se javiti neželjeni efekti kao što je istovremeno pristupanje zajedničkom resursu.




Nemoguće je utvriti vrednost promenljive *x* nakon izvršavanja BP-a. U 5 uzastopnih izvršavanja dobijaju se slučajne vrednosti: 107, -848, -192, 598, 415.

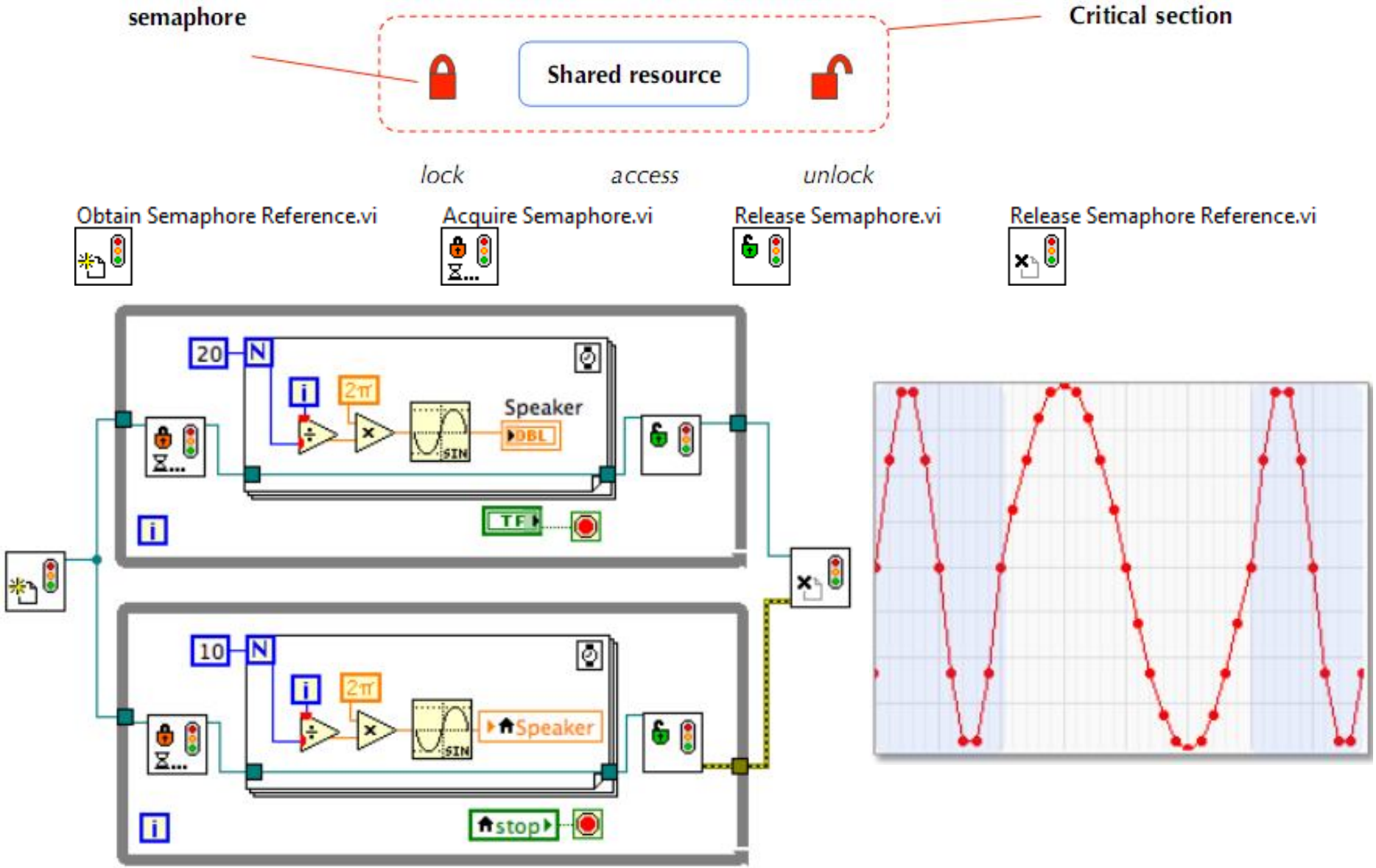
# Race condition



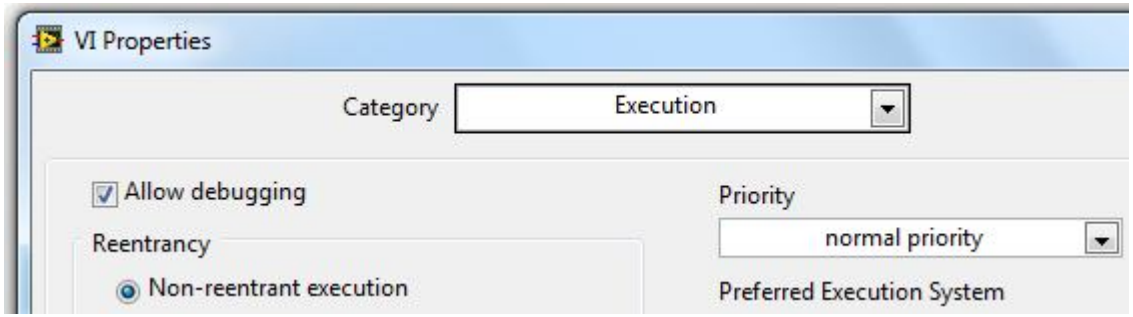
# *Race condition* – rešenje 1

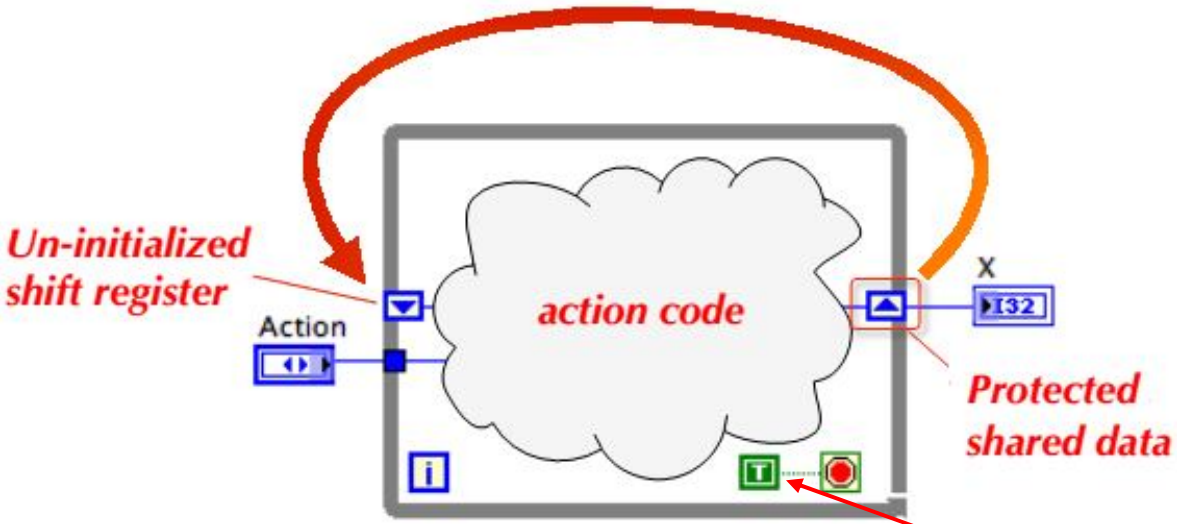
- Prvo moguće rešenje su semafori.



# Funkcionalne globalne promenljive

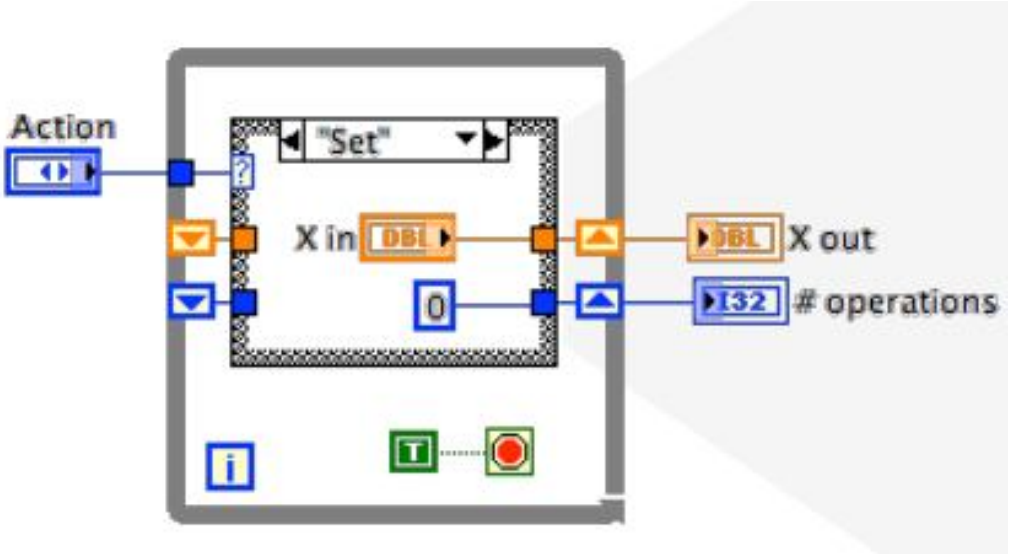
- Drugo moguće rešenje – Funkcionalne globalne promenljive.
- *FGV - Functional global variable*.
- Koristi se neinicialiozovani *Shift Register*.
- Korisnik definiše akciju koja će se izvršiti nad zaštićenim podatkom.
- VI mora biti *non-reentrant*.

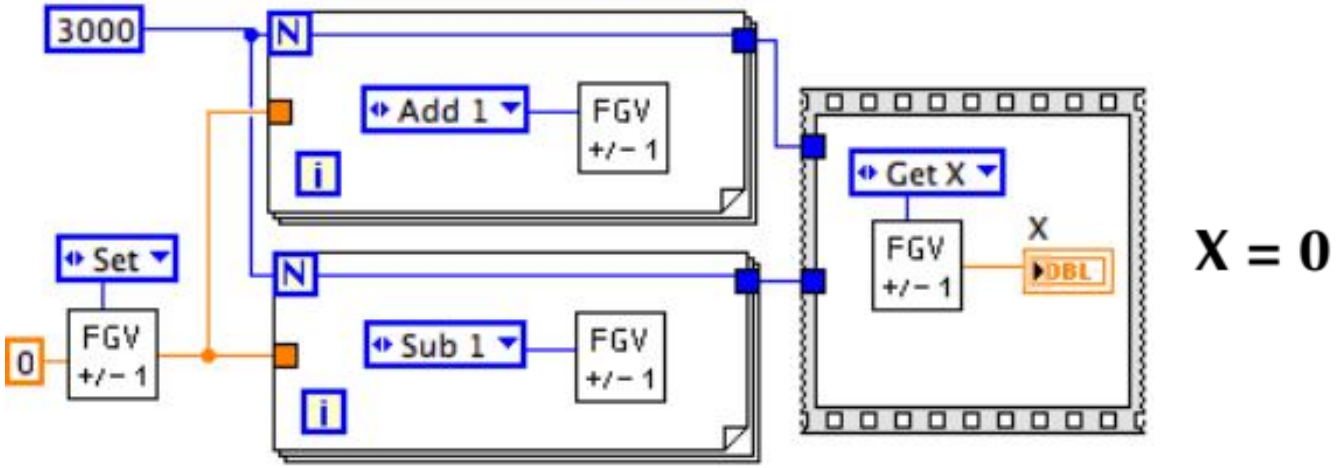


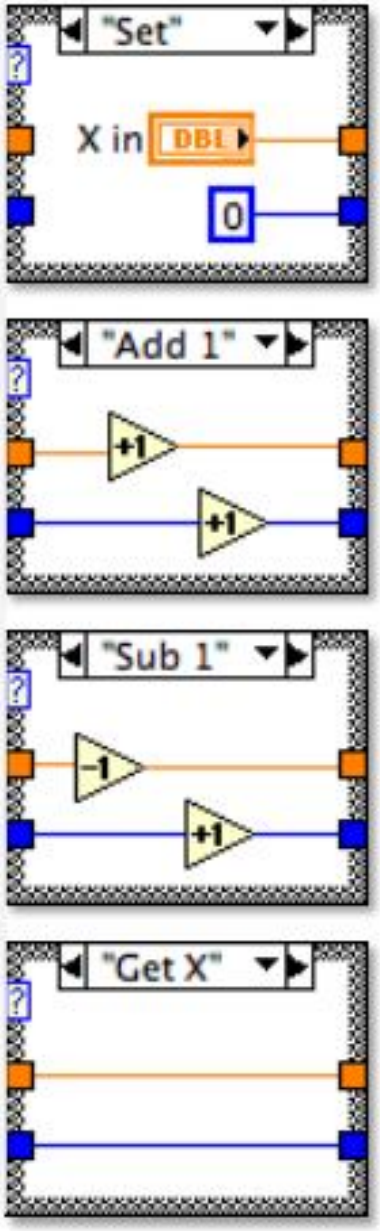

Izvršava se samo jednom

# Funkcionalne globalne promenljive



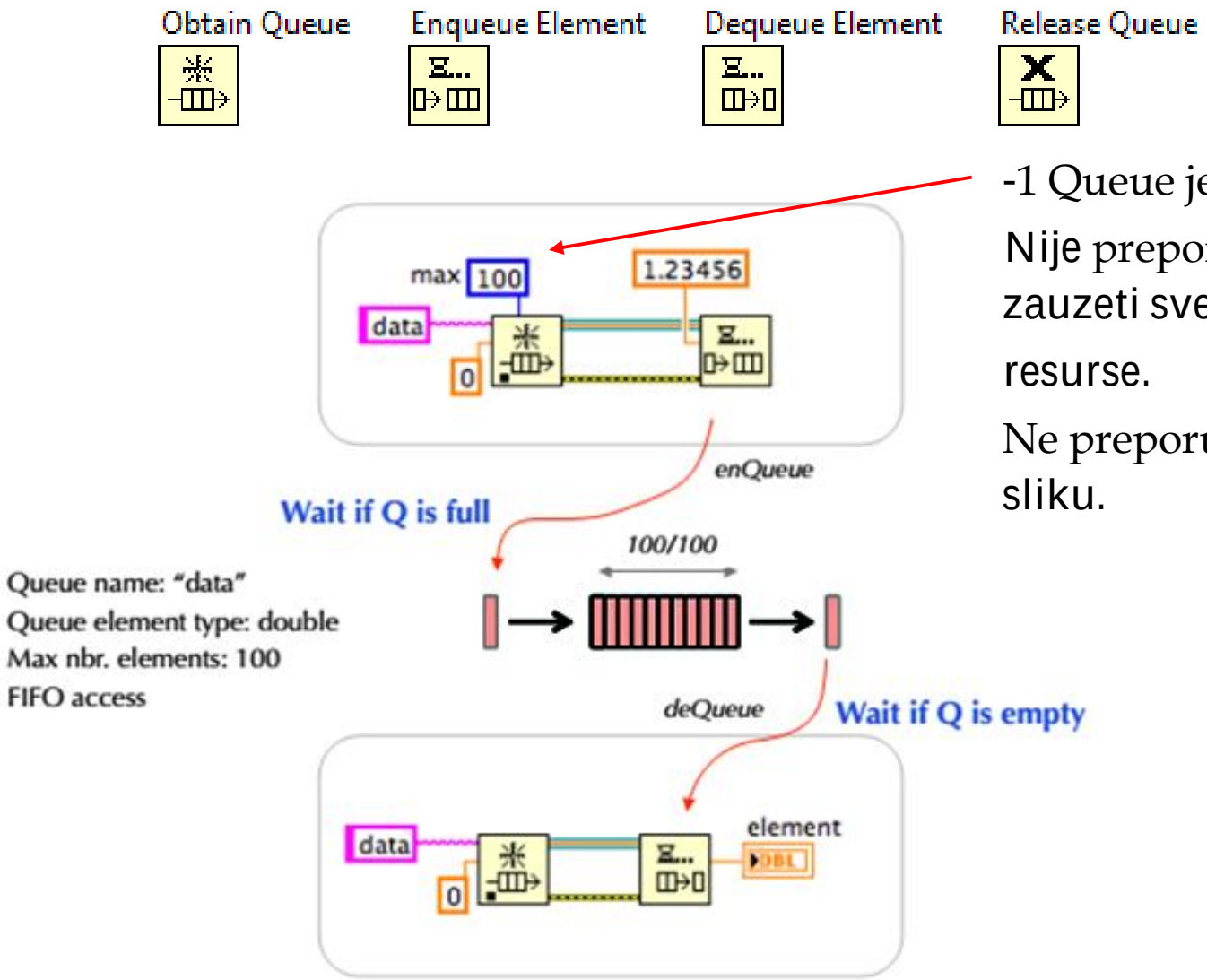




$X = 0$



# Queue

- Omogućavaju razmenu podata između dve petlje.

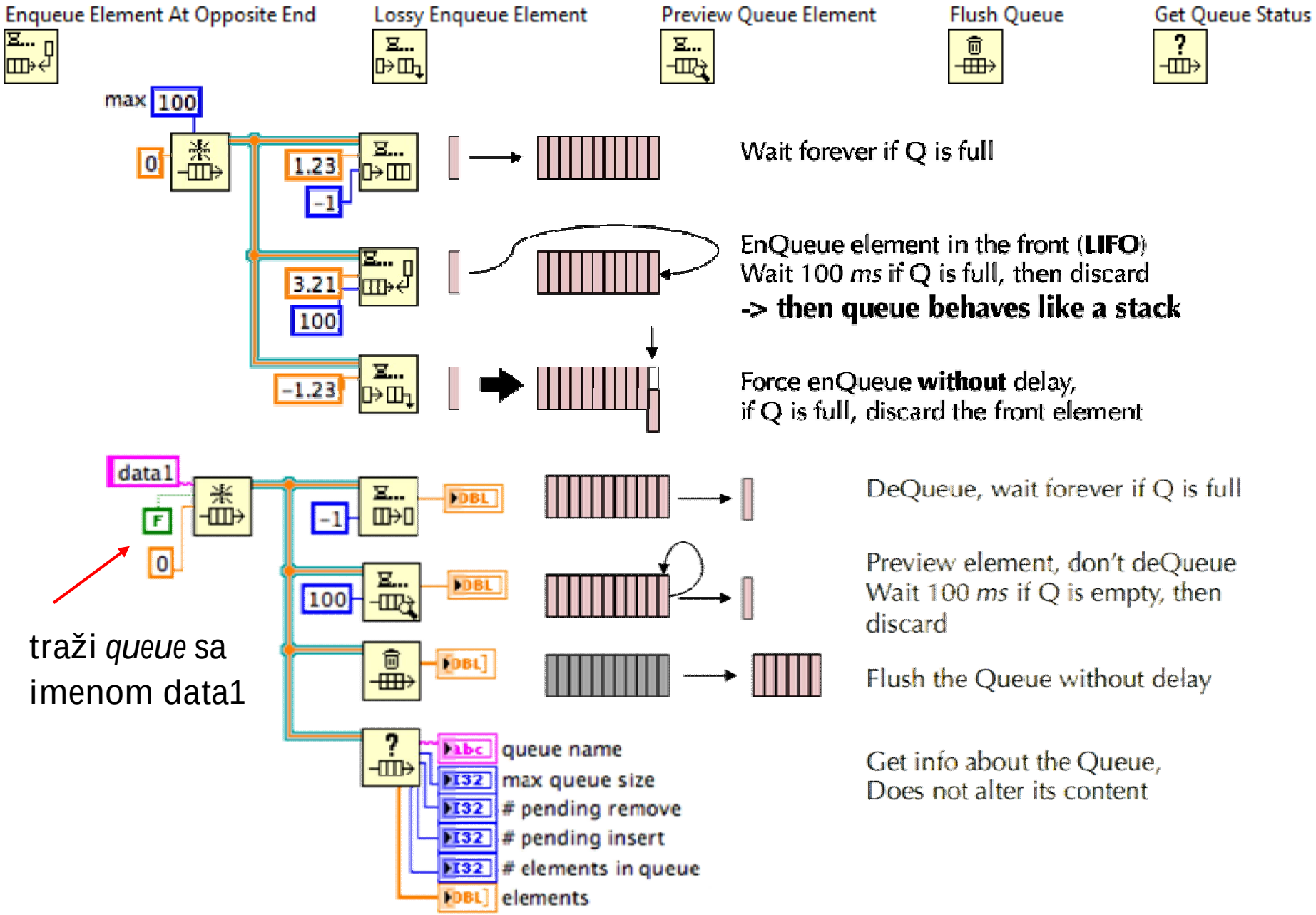



-1 Queue je neograničen.

Nije preporučljivo, može zauzeti sve memorijske resurse.

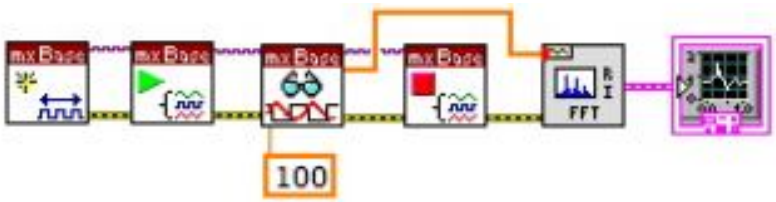
Ne preporučuje se za sliku.

# *Queue* – dodatne funkcije



## Design Patterns

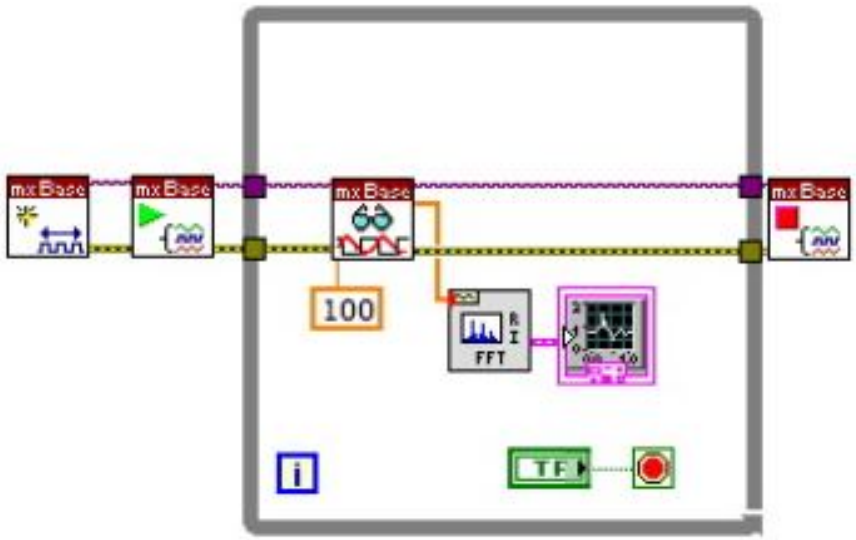
- *Design Patterns* ili programski šabloni su standardni način pisanja koda u LabVIEW koji se preporučuju:
  - *One shot,*
  - *One loop,*
  - *State machine,*
  - *Event loop* (*Event programing*),
  - *Multiple loops,*
  - *Producer/consumer.*




*One shot*

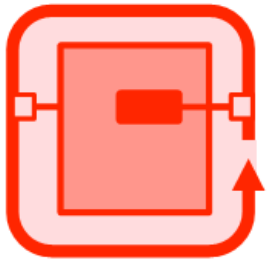
```
Config();
Start();
Acquire(100);
Stop();
PostProcess(FFT);
Display();
```

*One loop*

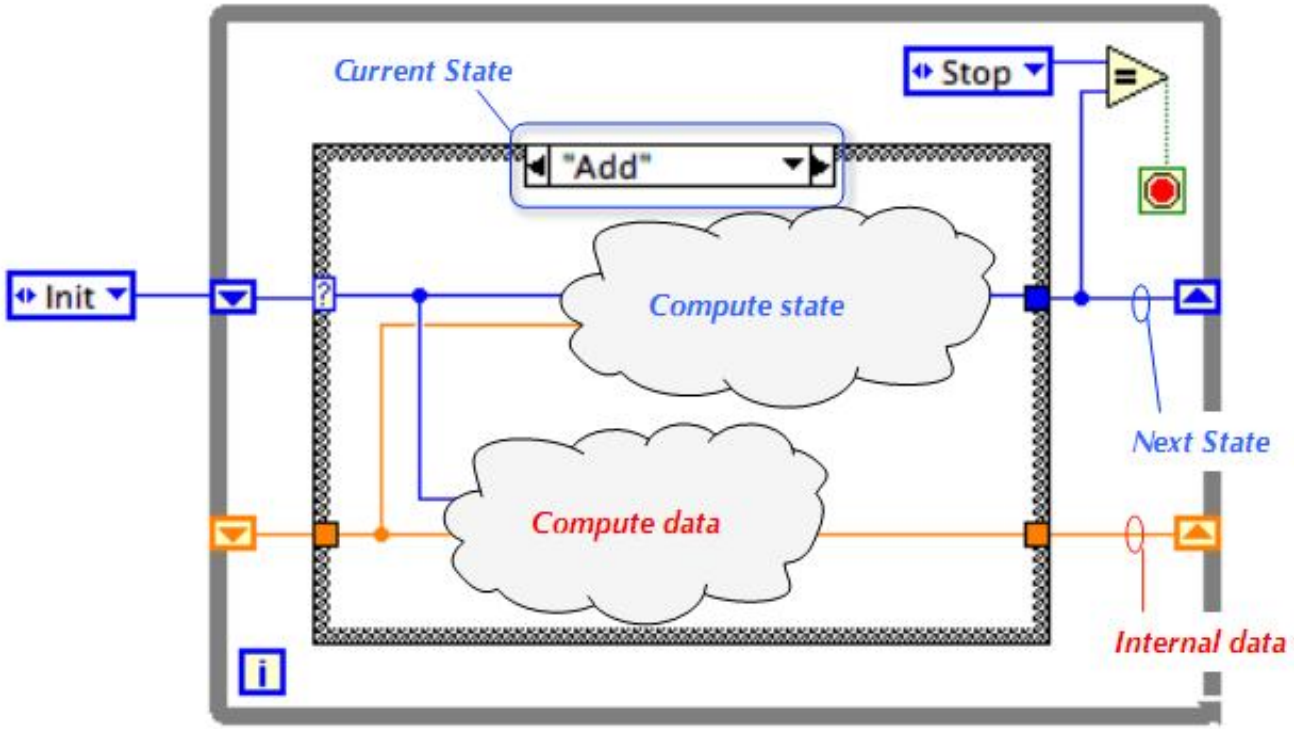




```
Config();
Start();
While (!Stop()) {
   Acquire(100);
   PostProcess(FFT);
   Display();
}
Stop();
```

# Design Patterns – State Machine



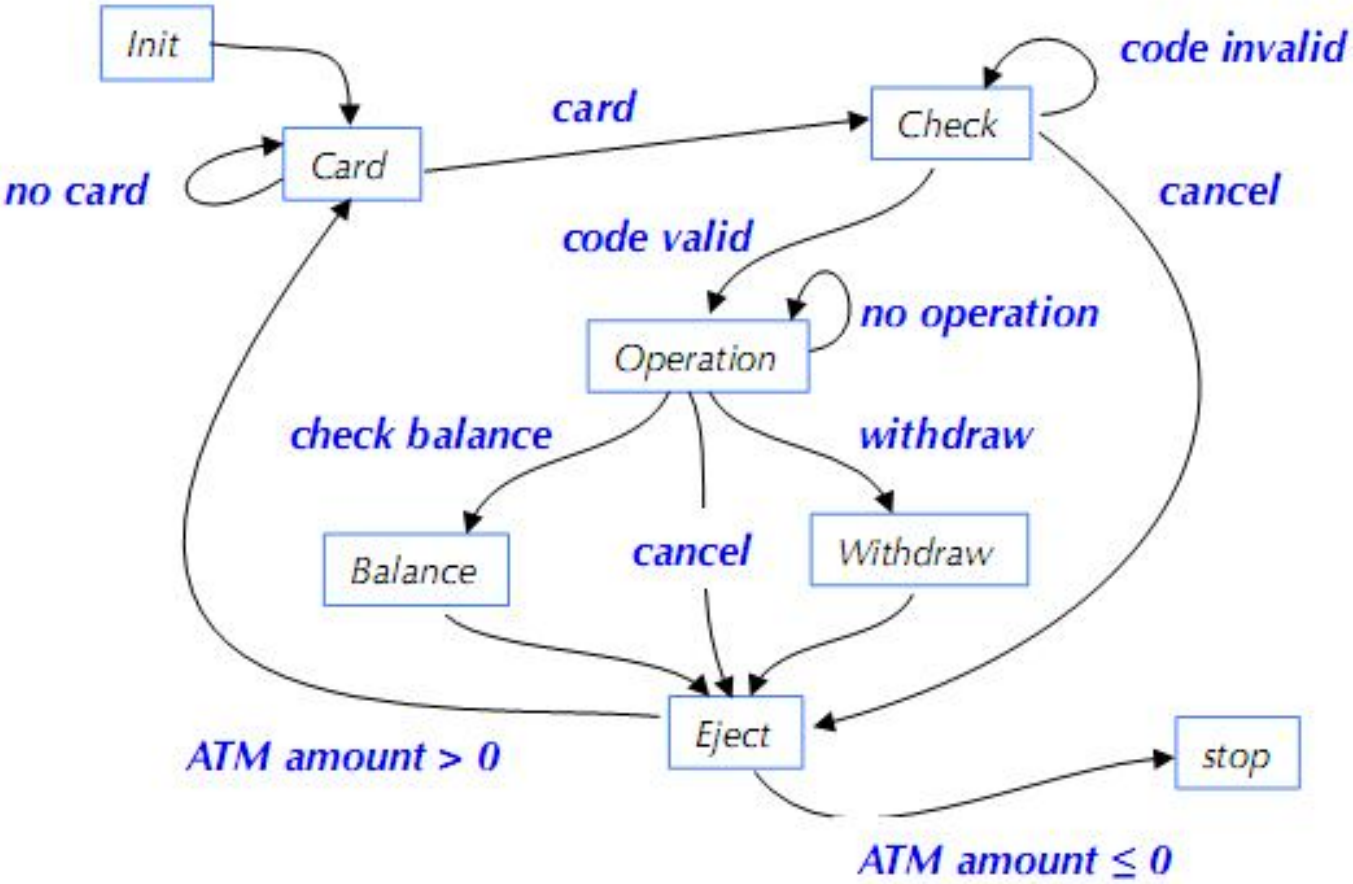
- *State Machine* – mašina stanja.
- Izvršava se sekvenca naredni, ali u odnosu na standardnu sekvencu, redosled sekvenci se određuje programski.
- Za promenu stanja koristi se *Shift Register*.
- *Case* struktura se koristi za svako stanje.
- U svakom stanju određuje se sledeće stanje.



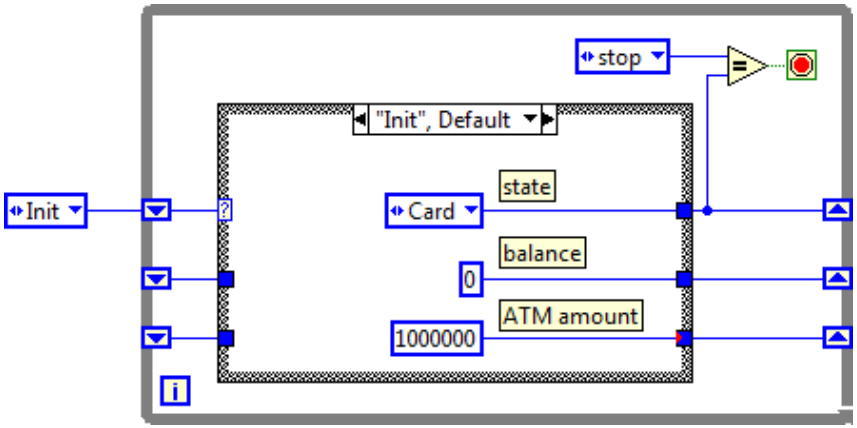

```
State = Init;
While (State!=Stop) {
  switch state {
    case Add: …
    case Sub: …
  }
  State = …
}
```

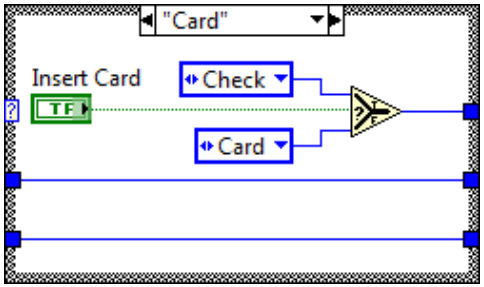
## Design Patterns – State Machine

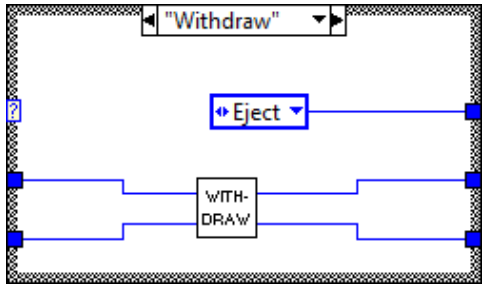
- Primer mašine stanja - ATM

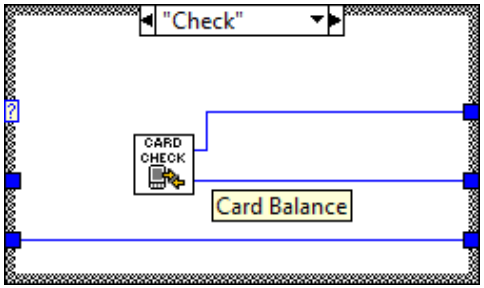


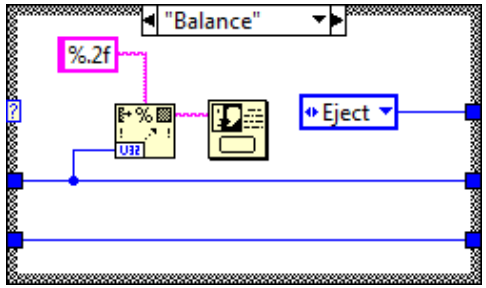
# Design Patterns – State Machine

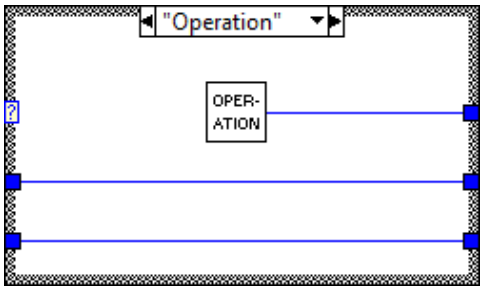


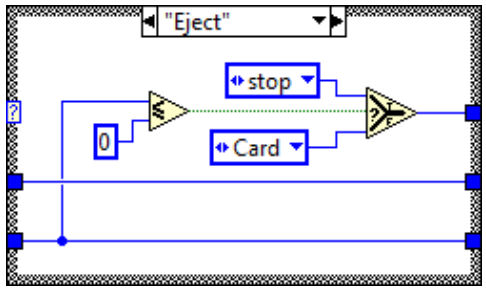



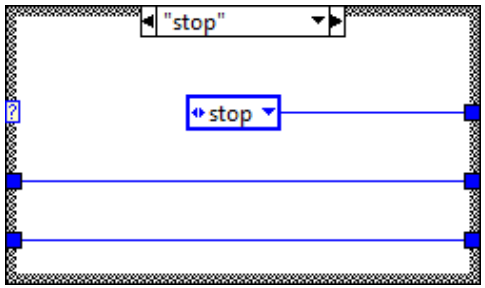



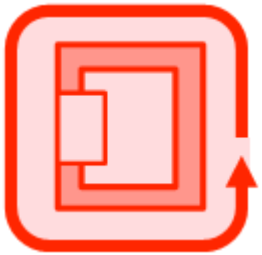



## Design Patters – Event Loop



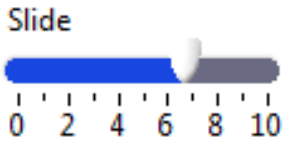
- *Event-driven programming* – omogućuva reakciju na interakciju korisnika sa korisničkim interfejsom (UI).
- *Events*, kao što su aktivacija tastera, klik miša, promena vrednosti kontrole registruje operativni sistem (OS) ili LabVIEW.
- Registrovane *events* "hvata" *Event* struktura (*Programming » Structures*) i prenosi odgovarajućem *case*-u. Takođe, presnose se i informacije o *event*-u.
- U slučaju pojave više *event*-ova, prvi se odmah obrađuje, a ostali se smešaju u red i obrađuju prema redosledu pojavljivanja.
- Moguće je "zaključati" UI sve dok se ne obradi tekući *event* (*default* podešavanje) .
- Čekanje na *event* ne opterećuje CPU.
- *Event* struktura omogućava filtriranje *event*-ova, npr. odbacivanje *Panel Close event*-a.
- Moguće je definisati dinamičke *event*-ove. Standardne *event*-ove može registrovati samo VI sa kojim korisnik "komunicira", dok ih ostali VI (pozvani kao SubVI) "ne vide". Dinamičke *event* "vidi" i SubVI.

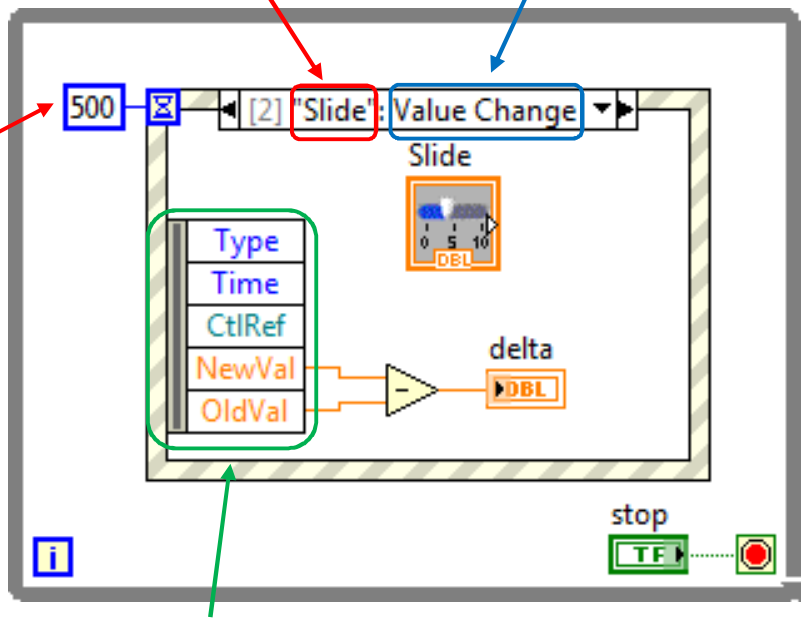
# Design Patters – Event Loop

Event source

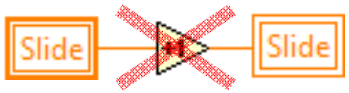
Registered event

Koliko *Event* struktura da čeka na *event* -1 za beskonačno.



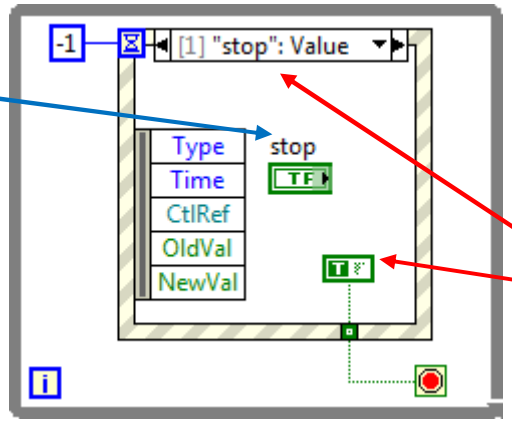




Event data node



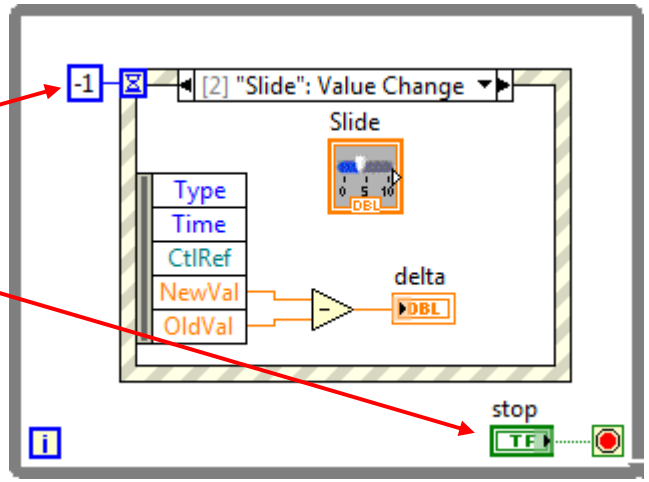

Izmena kontrole pomoću lokalne promenljive nije *Event Value Change*, jer je nije genirisao korisnik.

*Lacht* mora biti "pročitan" u odgovarajućem *case*-u, da bi mu se vrednost vratila na *false*.

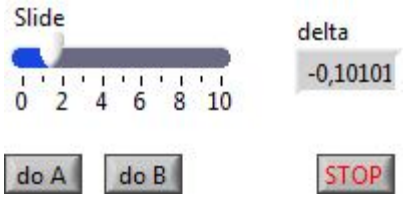


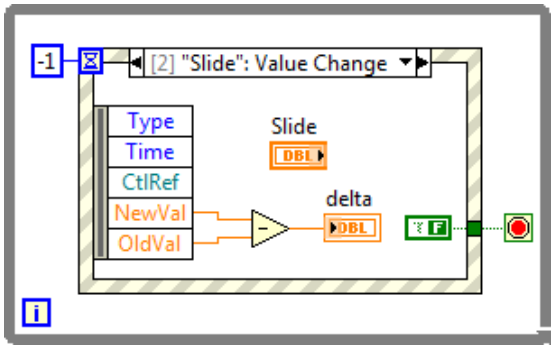

Petlja se ne prekida, jer se **stop** nikada ne izvršava.

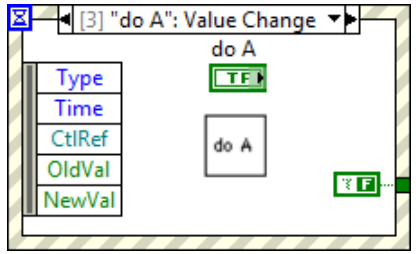
rešenje

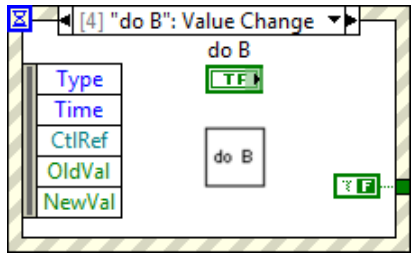


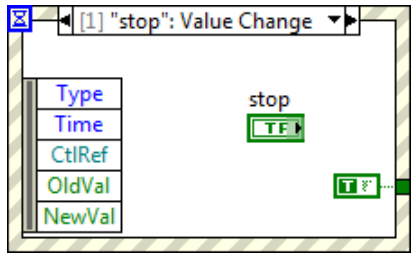
# Design Patters – Event Loop

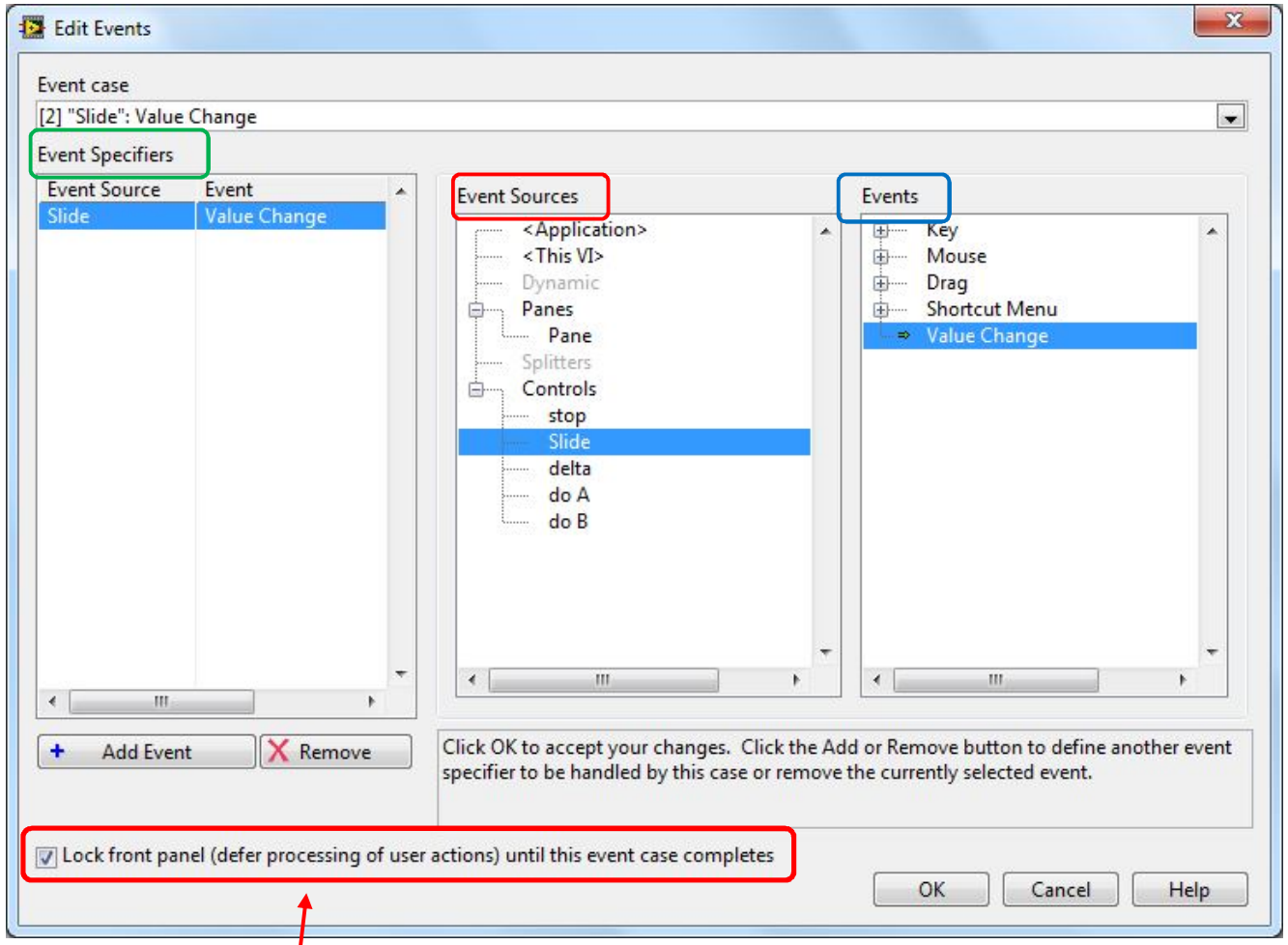


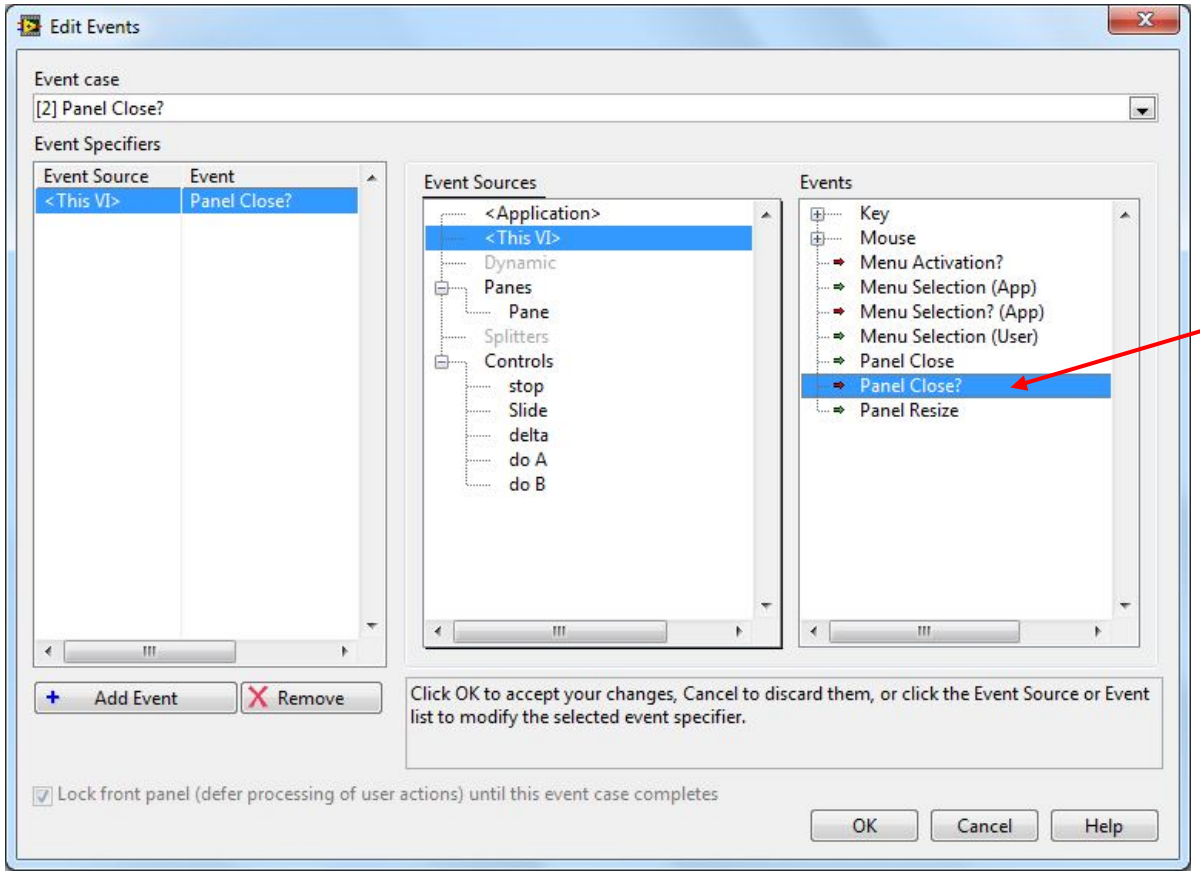



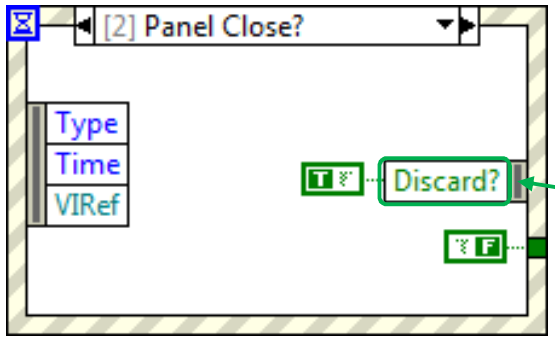

"Zaključavanje" FP-a (*default*).

# Design Patters – Event Loop



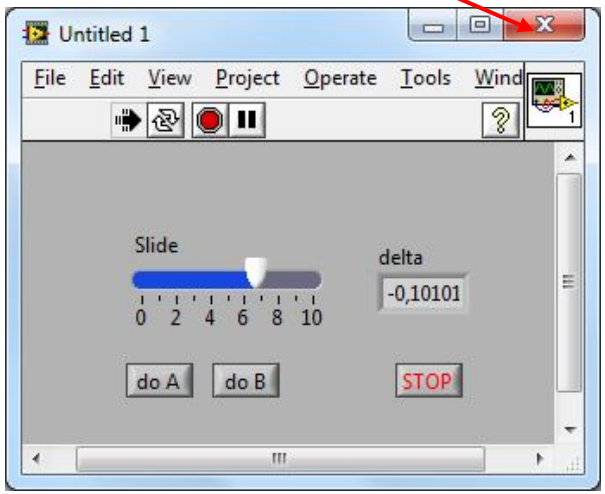

Filter *event* ima crvenu strelicu i znak pitanja.

➡ Panel Close?



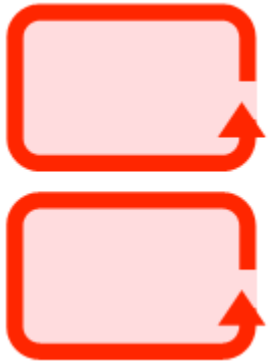

*Event filter node*

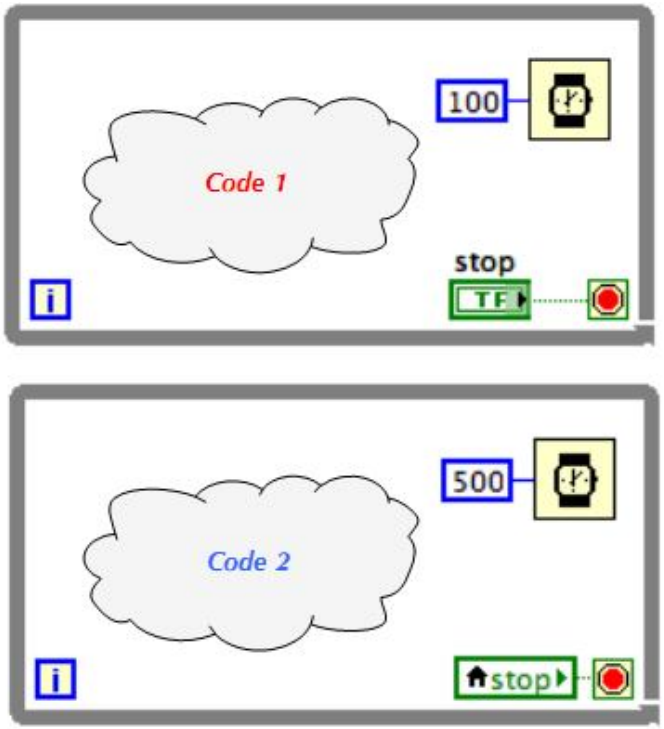
Sada ne reaguje na klik.



## Design Patterns – Multiple loops

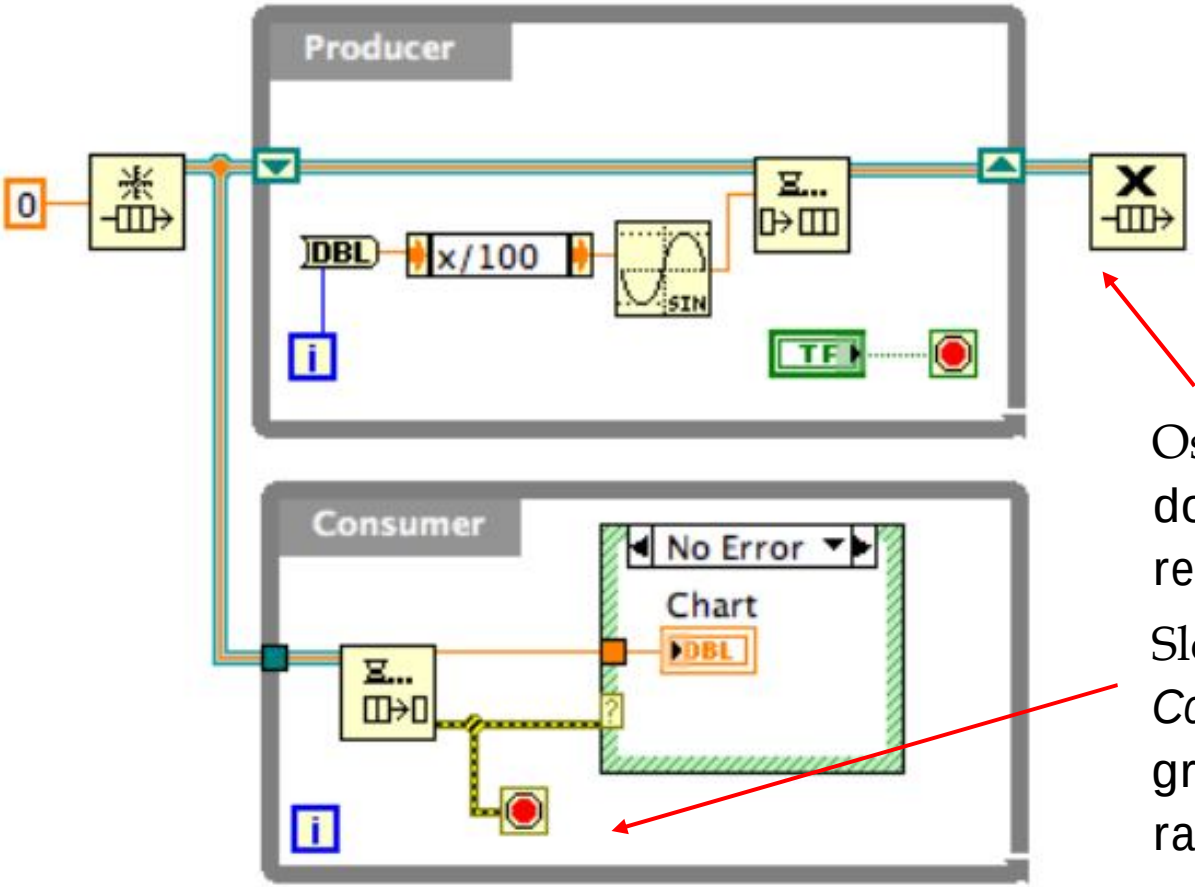
- Nezavisan, paralelan rad više petlji, koje ne moraju da dele resurse.
- Ukoliko petlje dele resurse, potrebno je koristi neki od mehanizama sprečavanja *Race Condition.*
- Petlje nisu sinhronizovane (postoji moućnost sinhornizacije korićenjem *subpalette Programming » Structures » Timed Sturctures*).
- LabVIEW, ukoliko postoje mogućnosti, svaku petlju dodeljuje drugom procesoru.





## Design Patterns – Producer / Consumer

- *Master* (*Producer*) generiše podatke i to može biti i asinhrono.
- *Slave* (*Consumer*) čeka da podaci budu dostupni, a zatim ih koristi.
- Razmena podataka između dve petlje se vrši pomoću *queue.*

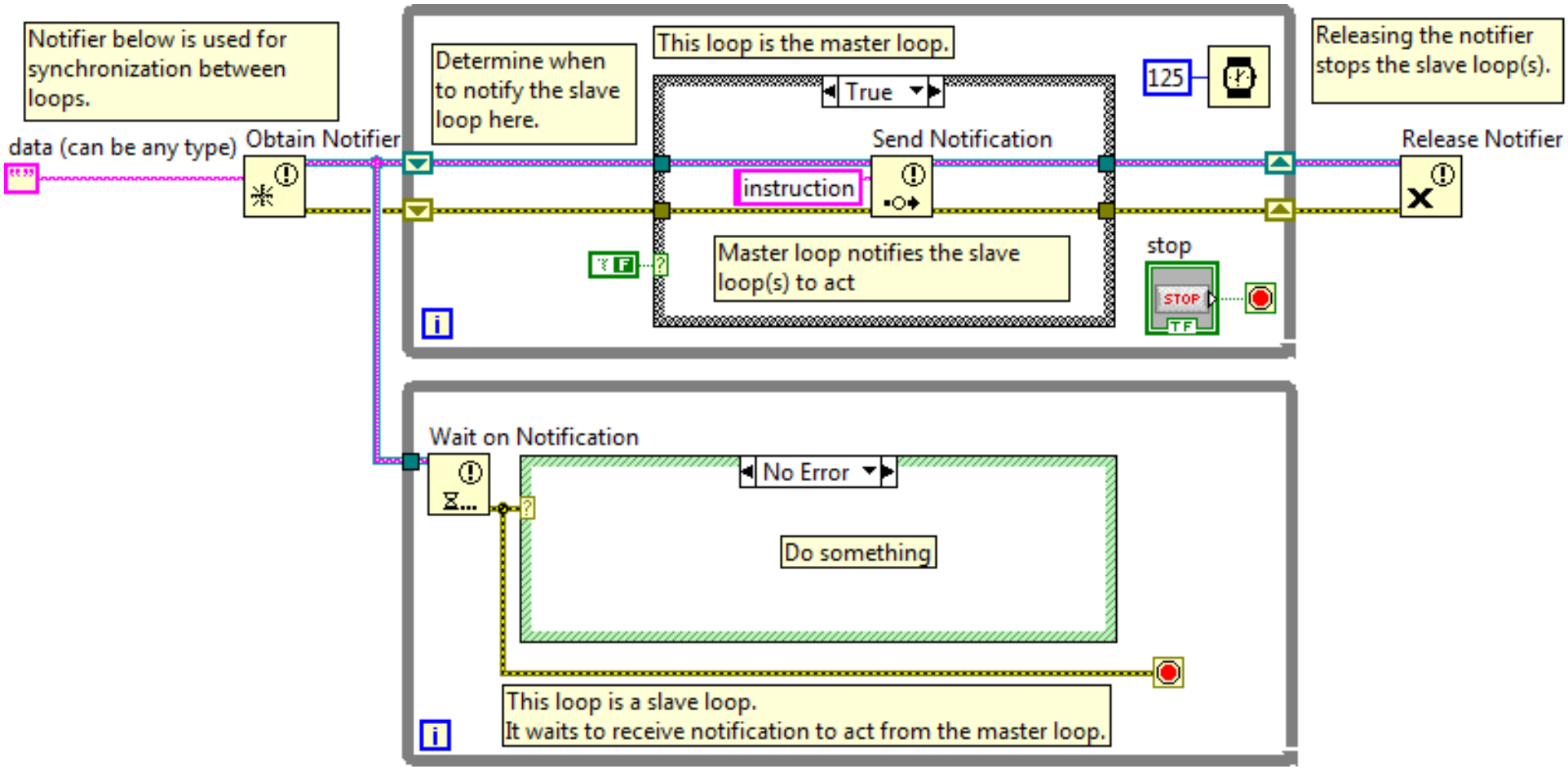



Oslobađa se memorija dodeljena *queue*, a time i referenca na *queue.*

Sledeći poziv *Dequeue* u *Consumer* petlji generiše grešku i ona prestaje sa radom.

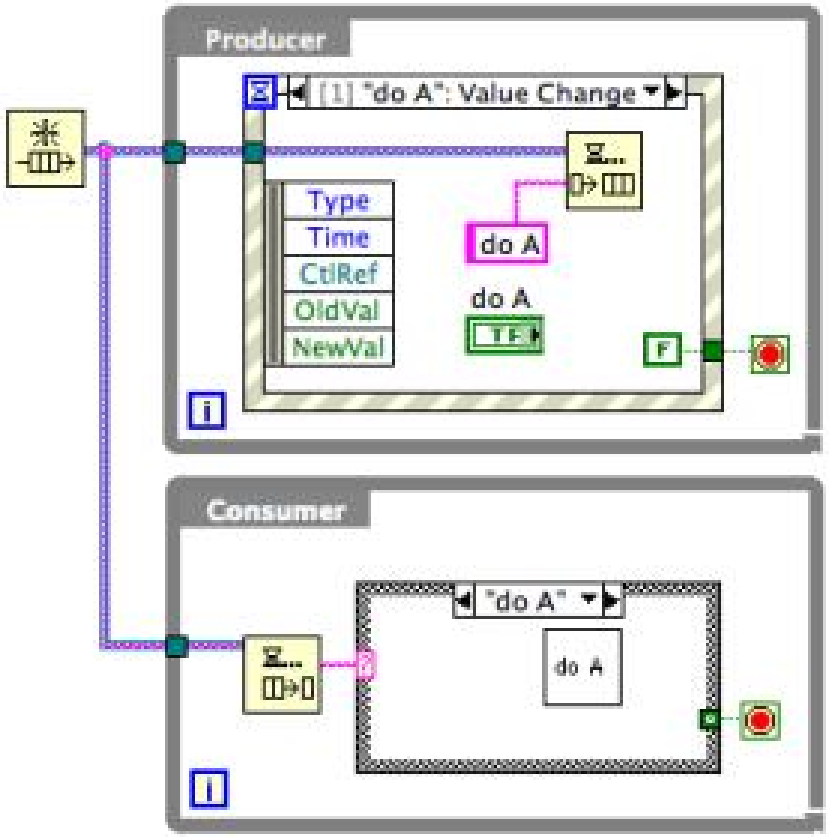
## Design Patterns – Producer / Consumer with Notifiers

- *Notifiers* – deo BD-a šalje poruku (koja može biti bilo koji tim podataka) drugom delu BD-a ili SubVI. Kada odredište primi poruku (Notification) nastavlja sa izvršavanjem programa.
- Teorijski mogao bi se sprečiti *Race Condition.*
- *Master/Slave* programski šablom se može realizovati pomoću *Notifiers.*



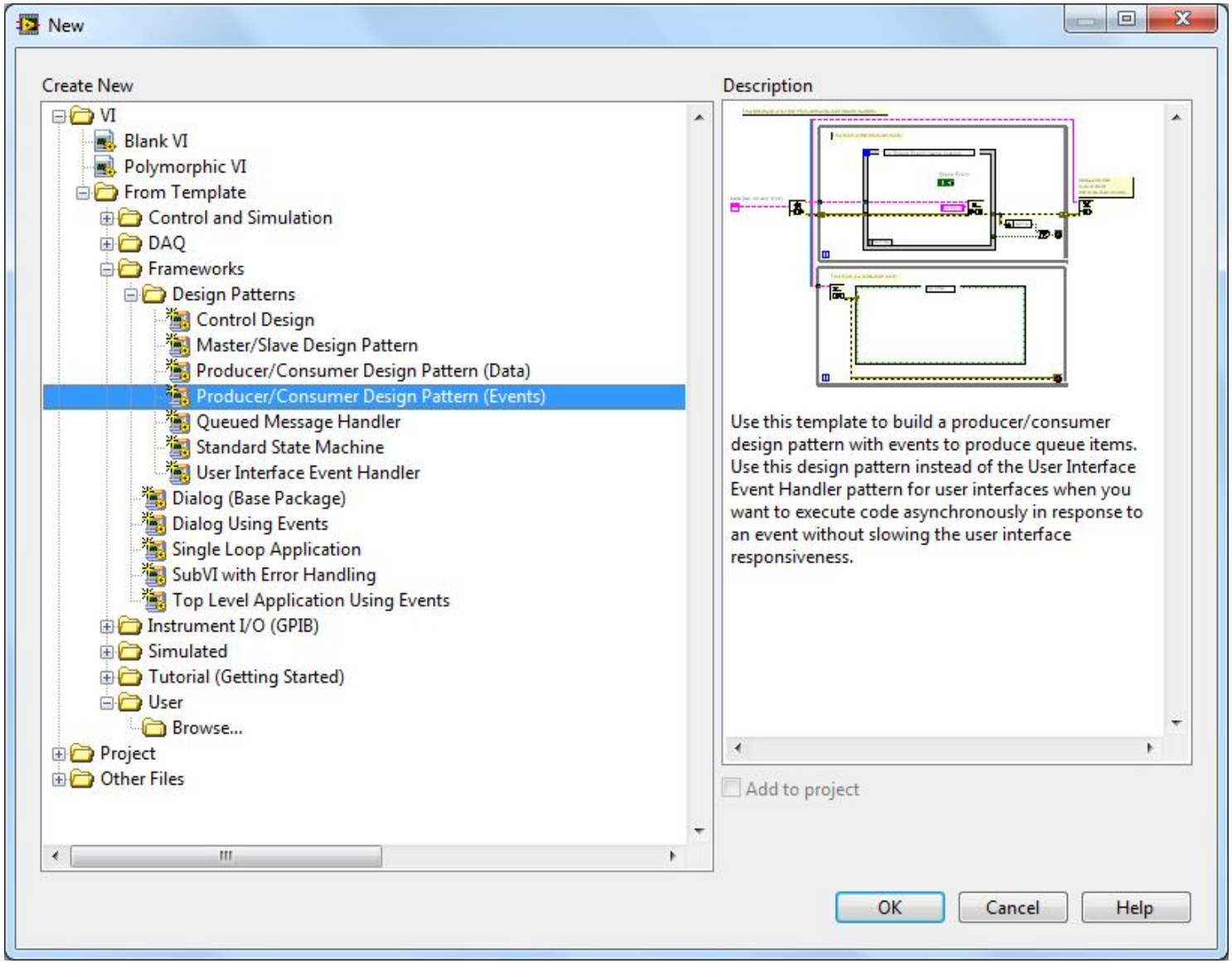
## Design Patterns – Producer / Consumer Event Loop

- *Producer* registruje *event* , ali ga ne obrađuje, već šale podatke o *event*-u *Consumer*-u pomoću *queue*-a.
- Registracija *event*-a vrlo kratko traje, sva obrada je odvija u *Consumer* petlji..



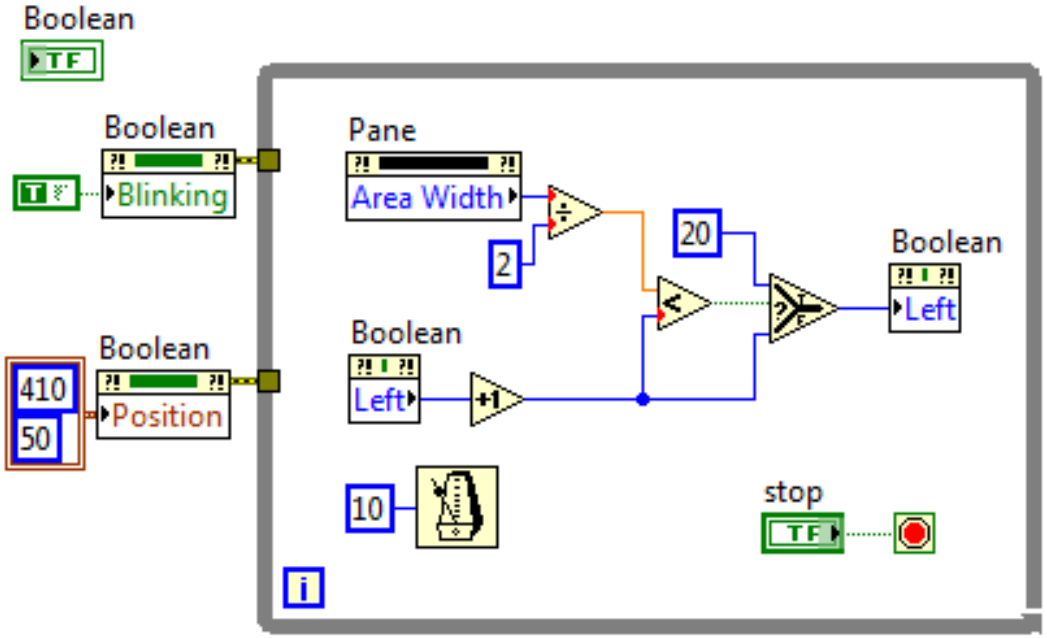
# Design Patterns

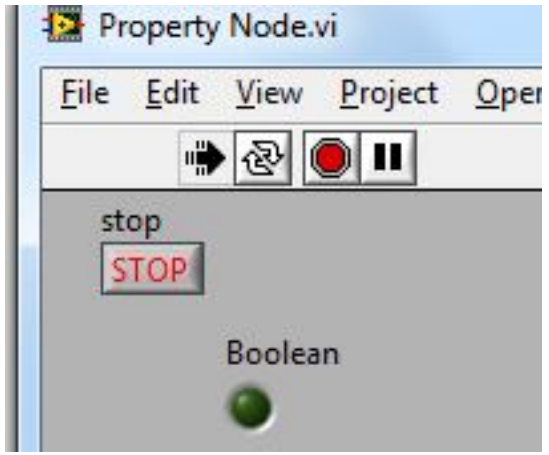
- *New VI from template.*



## Property Node

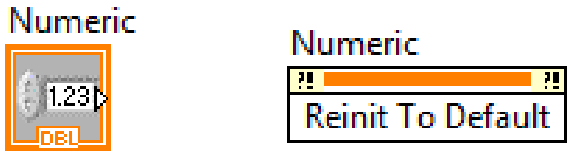
- Omogućava izmenu izgleda FP u toku izvršavanja programa, npr. određenim korisnicima je dozvoljen pristum samo nekim kontrolama, ostale moraju biti u stanju *Disabled.*
- Desni klik na terminal (BD) » *create* » *property node*, a zatim izbor *property-a* kojem se želi pristupiti.
- Ne podržavaju svi property funkciju *write,* već samo *read.*
- *Property Node* može biti povezan i sa delom FP-om u koji se mogu smestiti kontrole / indikatori (*Pane*) .



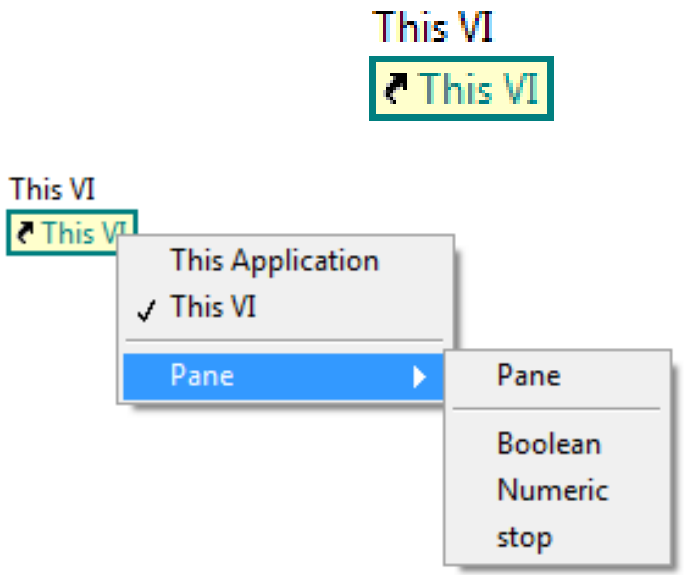



## *Invoke Node* i *Control Reference*

- Koristi se za izvršavanje neke akcije na objektu u toku izvršavanja programa.
- Za razliku od *property* koji predstavlja neku o osobina obekata, *invoke node* je metod, tj. operacija koja se može izršiti nad objektom.
- *Invoke Node* se pristupa desnim klikom » *create* » *invoke node* i izbor metode koja se želi izvršiti.

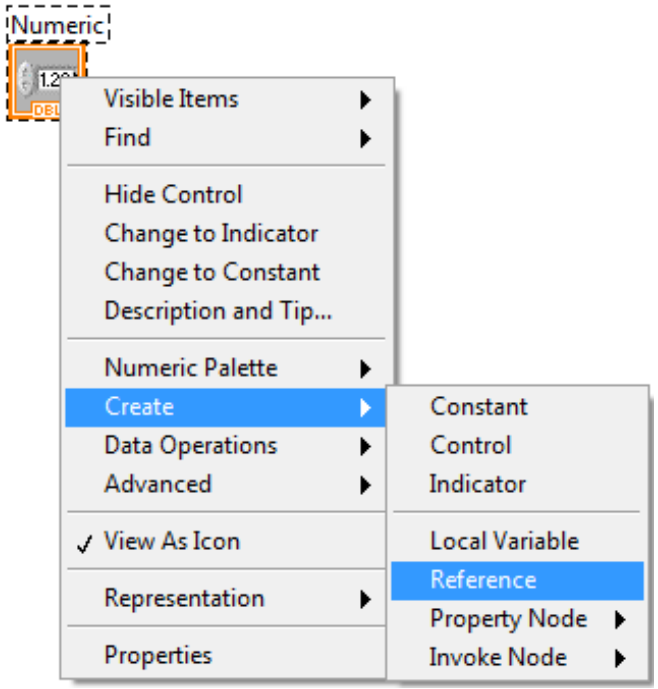



- *Control Reference* – pokazivač na objekat LabVIEW-a. Omogućava i referenciranje objekata koji se nalaze u drugom VI.
- Može se još referencirati i aplikacija (LabVIEW) , tekući VI (*This VI*) i sam panel (Pane) tekućeg VI.
- Paleta *Programming* » *Application Control* i izbor *VI Server Reference*. Zatim klik na Referencu i moguće je izbrati objekat sa kojim će se povezati referenca.

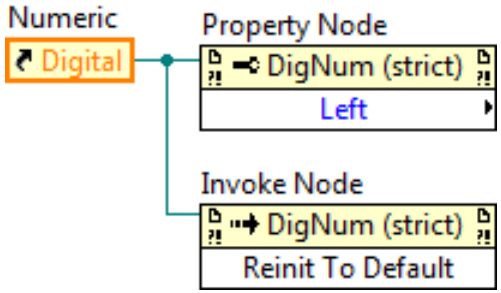


# *Property Node, Invode Node* i *Control Reference*

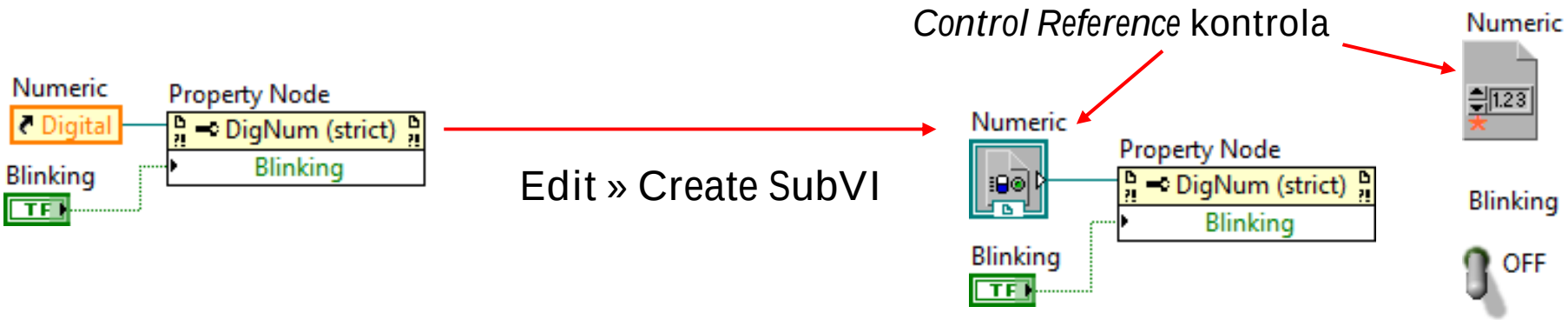
- *Control Reference* za objekat FP-a se može dobiti i desnim klikom i izborom *Reference*.



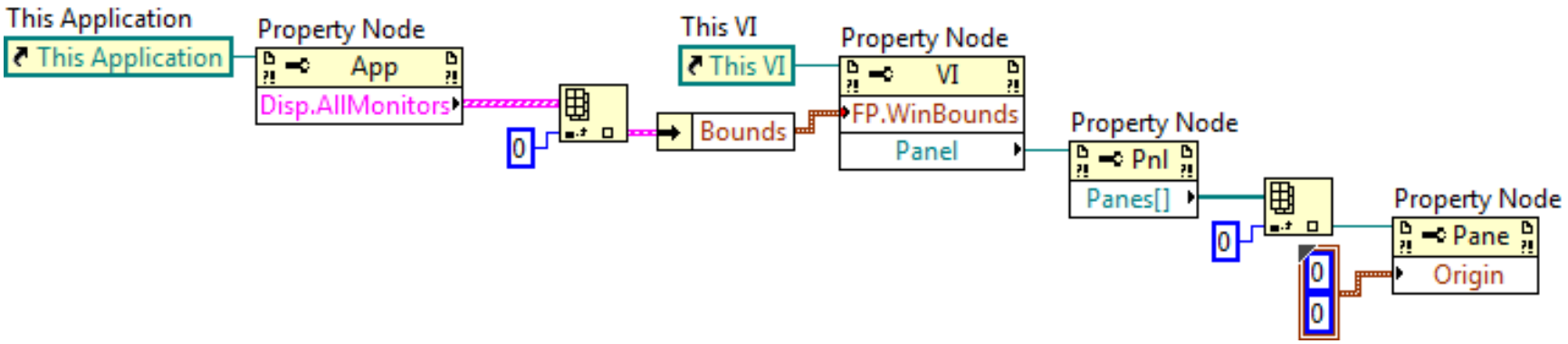

- *Control Reference* omogućava pristum *Property*-ima i *Method*-ama odgovarajućeg objekta pomoću *Property Node* i *Invoke Node* iz palete *Programming » Application Control.*

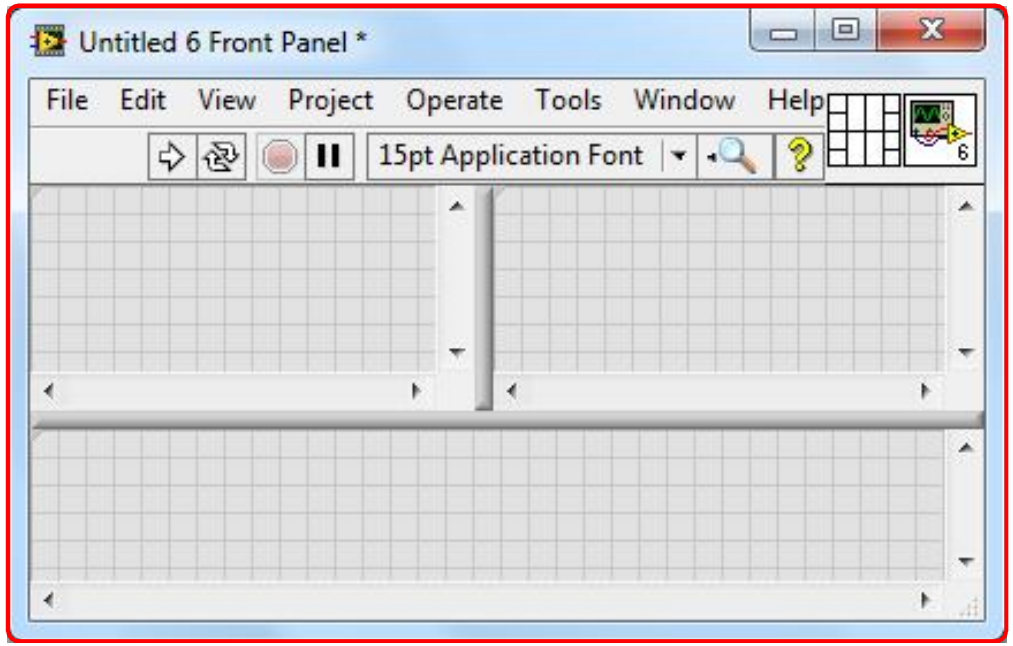



- *Control Reference* omogućava izmenu *Property*-a iz subVI.



# Property Node, Invode Node i Control Reference



# *Property Node, Invode Node* i *Control Reference*

- *Default* – FP ima jedan *Pane*, a *Origin* (levi gorni ugao)