VORLESUNG

Automatisierung industrieller Workflows

Teil C: Die Sprache SLX

- DISCO- Elemente -

Joachim Fischer

Inhalt C.6

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  ODEMx-Elemente

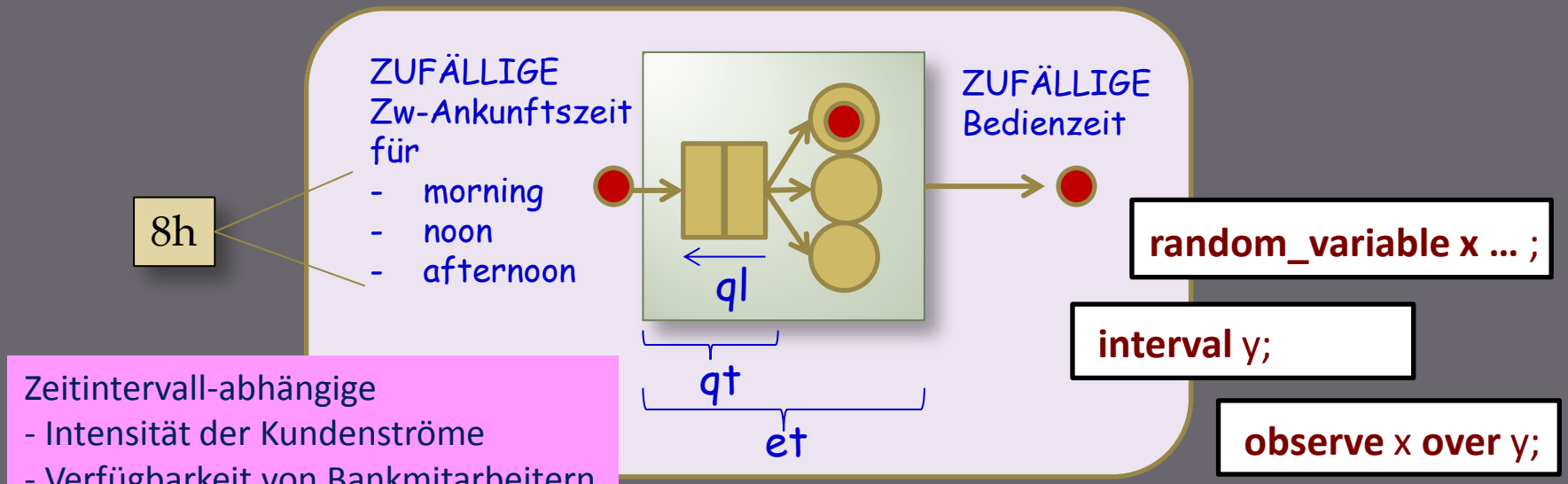
Schwerpunkte der letzten Vorlesung

Obektorientierung

- **C.6**
  DISCO-Elemente

1. Simula-Bibliothek DISCO
2. SLX-Bibliothek Statistics
   - Modellgrößen, Bewertungsgrößen, Kennwertermittlung
   - Random_Variable, Histogram, Statistics
   - Einfache Simulationsläufe, Histogramm-Auswertung
   - Zeitintervall-basierte Beobachtung und Auswertung
   - Konfidenz-Intervall-Schätzung
   - Sequential Sampling
   - CI-Schätzung mit antithetischen Läufen
   - Vergleich zwischen simulierten Systemvarianten

Zufallsgrößen und Intervall-Statistik (Wdh.)

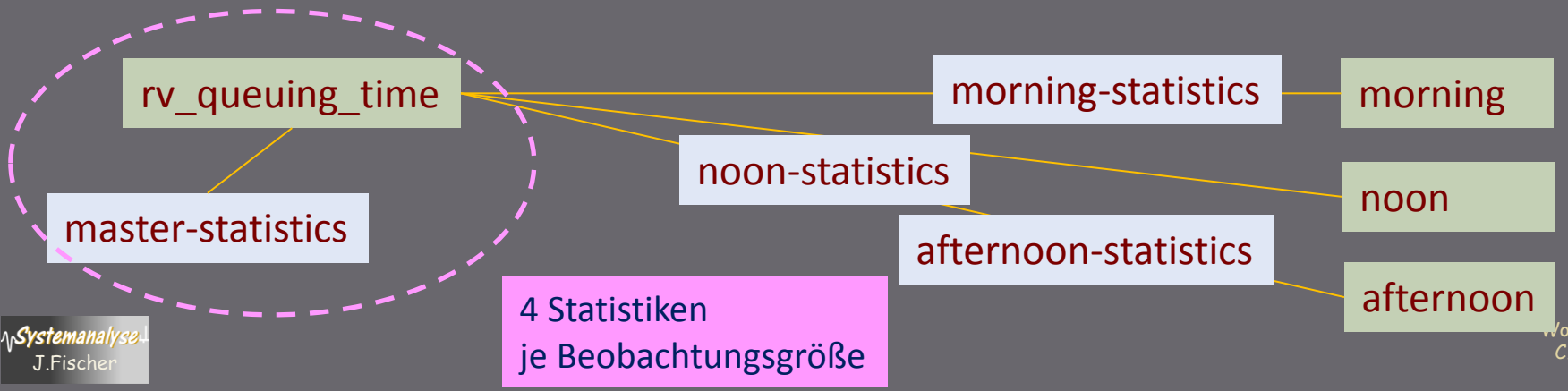


Zeitintervall-abhängige
- Intensität der Kundenströme
- Verfügbarkeit von Bankmitarbeitern

random_variable- Instanzen

statistics- Instanzen

interval- Instanzen



4 Statistiken
je Beobachtungsgröße

Idee zur Umsetzung (Wdh.)

```
procedure run_model () {
  float intensity=2.0;
  fork  {     // Arriving Customer
        forever  {
                advance rv_expo ( arrive , 1/intensity );
                activate new cl_customer;
                if ( door_closed ) terminate;
         }
   }
   fork  {   // Controlling the bank
    /* Morning Time */
        start_interval morning;
        intensity = 3 ;
        advance 3*60 ;
        stop_interval morning;
   /* Noon Time */
        enter clerk units=2;   //Entzug von Schaltern, evtl. Verzögerung
                               // des Beginns der Mittagszeit
        start_interval noon;
        intensity = 2 ;
        advance 2*60 ;
        stop_interval noon;
        leave clerk units=2 ;
   /* Afternoon Time */
       start_interval afternoon;
       intensity = 3;
       advance 3*60;
       stop_interval afternoon;
       door_closed = TRUE;
       terminate;
  }
  wait until ( (time > close_time) && ( in_customer == out_customer ));
  report ( system );
}
}
```

- Intervall-abhängiger Eingabeparameter

Änderung der Intensität

Änderung der Verfügbarkeit von Bankangestellten
main-Puck belegt Storage neben den Kunden

<u>ACHTUNG</u>
Hier wird die Auslastung des Storage-Elementes verfälscht

```
//*************************************************************************
// Example EX0023
//*************************************************************************
import <stats>
import <h7>
module basic {
      rn_stream arrive, service ;
      random_variable rv_elapsed_time,
                  rv_queueing_time
                  histogram start=0.0 width=0.5 count = 20;
      random_variable (time) rv_quelength;
      storage clerk capacity=5;
      interval morning, noon , afternoon ;
      control int in_customer, out_customer;
      int que_length;
      constant float close_time=8*60, service_time=1.3 ;
      boolean door_closed;
class cl_customer {
       actions {
                  in_customer ++; // increment customer counter
                  que_length++; // increment queue length
                  tabulate rv_quelength=que_length; // tabulate
                  enter clerk; // try to catch a clerk
                  que_length --; // decrement queue length
                  tabulate rv_quelength=que_length count=0 ;
                  tabulate rv_queueing_time= time - ACTIVE->mark_time;
                  advance rv_expo ( service , service_time ); // service time
                  leave clerk;
                  out_customer ++;
                  tabulate rv_elapsed_time = time - ACTIVE->mark_time;
       }
}
procedure main() {
                  observe rv_elapsed_time, rv_queueing_time , rv_quelength over
                  morning, noon , afternoon ;
                  run_model();
}// main
```

```
procedure run_model () {
   float intensity=2.0;
   fork  {     // Arriving Customer
         forever  {
                  advance rv_expo ( arrive , 1/intensity );
                  activate new cl_customer;
                  if ( door_closed ) terminate;
         }
   }
   fork  {   // Controlling the bank
    /* Morning Time */
         start_interval morning;
         intensity = 3 ;
         advance 3*60 ;
         stop_interval morning;
   /* Noon Time */
         enter clerk units=2;   //Entzug von Schaltern, evtl. Verzögerung
                                // des Beginns der Mittagszeit
         start_interval noon;
         intensity = 2 ;
         advance 2*60 ;
         stop_interval noon;
         leave clerk units=2 ;
   /* Afternoon Time */
         start_interval afternoon;
         intensity = 3;
         advance 3*60;
         stop_interval afternoon;
         door_closed = TRUE;
         terminate;
   }
   wait until ( (time > close_time) && ( in_customer == out_customer ));
   report ( system );
}
}
```

EX-0023 (Bank, Interval).slx: SLX-64 OV031 Lines: 7,507 Errors: 0 Warnings: 0 Lines/Second: 292,008 Memory: 4 MB

Execution begins

System Status at Time 481.8975

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrive	1263	200000	201263	OFF	0.38
service	1263	400000	401263	OFF	0.33

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
rv_elapsed_time	1263	1.870	1.656		0.01	12.93
rv_quelength	1263	1.476	2.565		0.00	13.00
rv_queueing_time	1263	0.563	0.978		0.00	5.36

Lower	Upper	Frequency	Percent	
0.0	0.5	872	69.042	| **********
0.5	1.0	123	9.739	| *******
1.0	1.5	110	8.709	| ******
1.5	2.0	49	3.880	| **
2.0	2.5	43	3.405	| **
2.5	3.0	14	1.108	|
3.0	3.5	10	0.792	|
3.5	4.0	8	0.633	|
4.0	4.5	25	1.979	| *
4.5	5.0	8	0.633	|
5.0	5.5	1	0.079	|

Master-Statistik

3. Intervall

Statistics Collection Intervals

Interval	Elapsed Time
afternoon	180.0000

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
rv_elapsed_time	507	1.687	1.484		0.01	12.93
rv_queueing_time	511	0.385	0.590		0.00	2.57

Lower	Upper	Frequency	Percent	
0.0	0.5	364	71.233	| **********
0.5	1.0	53	10.372	| *******
1.0	1.5	61	11.937	| ********
1.5	2.0	22	4.305	| ***
2.0	2.5	10	1.957	| *
2.5	3.0	1	0.196	|

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
rv_quelength	511	1.094	1.951		0.00	11.00

Inhalt C.6

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  ODEMx-Elemente
- Obektorientierung
- **C.6**
  DISCO-Elemente

Schwerpunkte der letzten Vorlesung

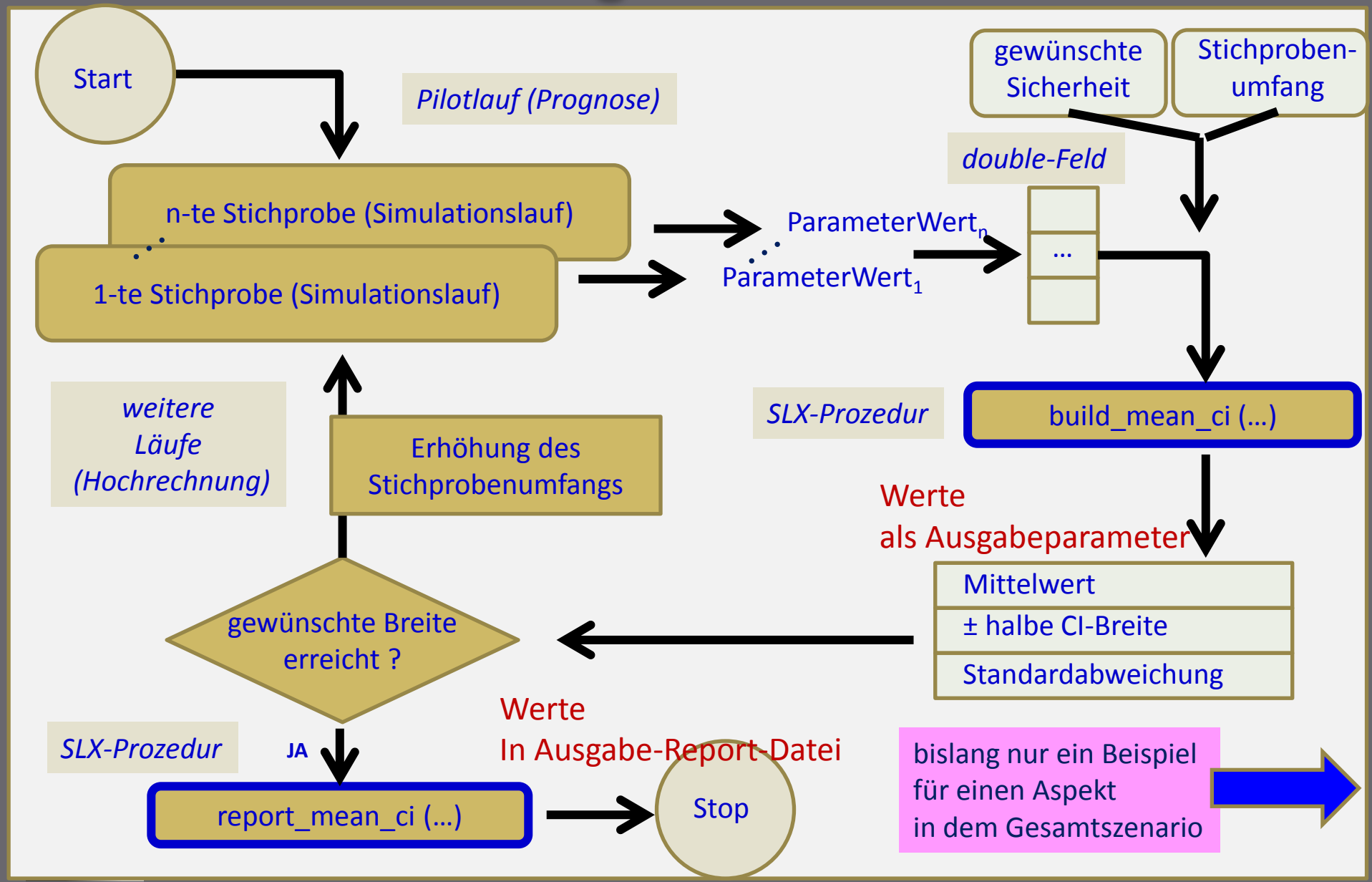
1. Simula-Bibliothek DISCO
2. SLX-Bibliothek Statistics
   - Modellgrößen, Bewertungsgrößen, Kennwertermittlung
   - Random_Variable, Histogram, Statistics
   - Einfache Simulationsläufe, Histogramm-Auswertung
   - Zeitintervall-basierte Beobachtung und Auswertung
   - Konfidenz-Intervall-Schätzung
   - Sequential Sampling
   - CI-Schätzung mit antithetischen Läufen
   - Vergleich zwischen simulierten Systemvarianten

Globale SLX-Prozeduren

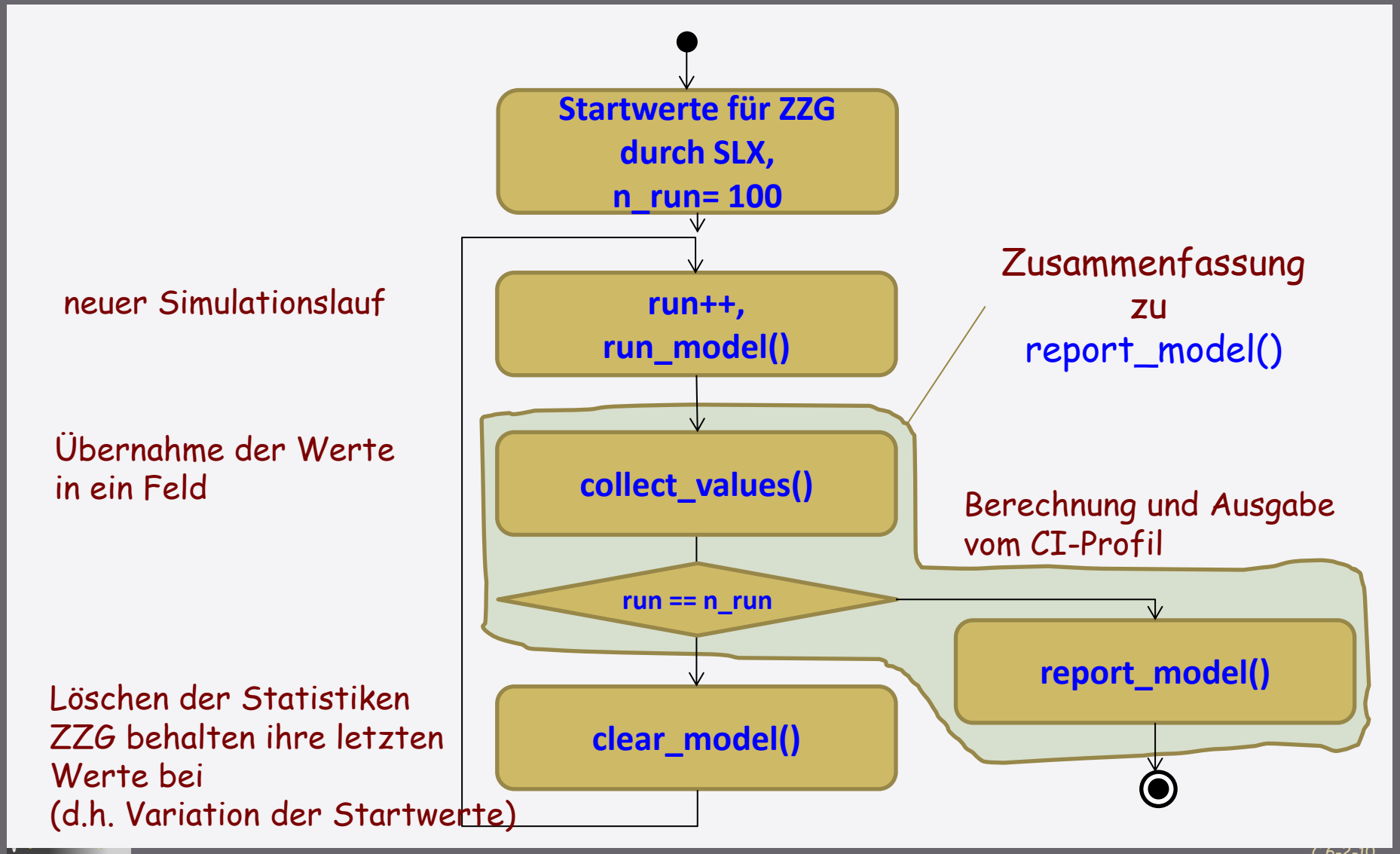
- **procedure** build_mean_ci
  (samples[*], scount, level, smean, stdev, half_width)

- **procedure** report_mean_ci
  (title, level, samples[*], scount)

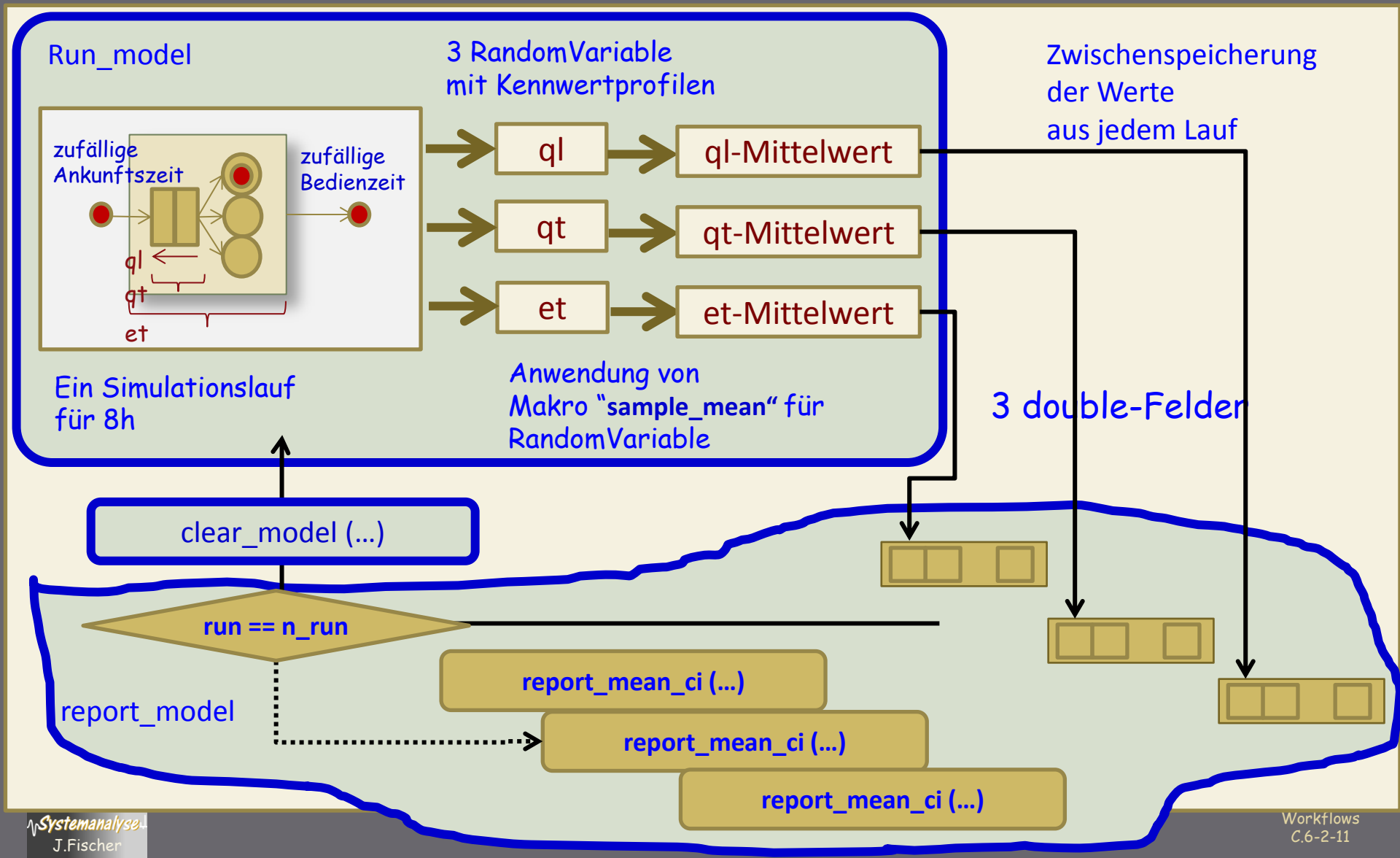
CI-Bestimmung in SLX: Datenfluss



Beispiel: Berechnung von Ci nach n Simulationsläufen



run_model(), report_model(), clear_model()



Beispiel: Ergebnis

Confidence Intervals after 100 runs

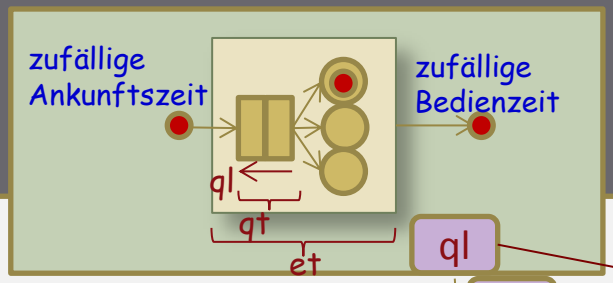
Konfindenzintervall

Elapsed Time				
Replications: 100	95% C.I.	3.5150	+- 0.2167	Std Dev: 1.0920
Queueing Time				
Replications: 100	95% C.I.	2.2204	+- 0.2107	Std Dev: 1.0619
Average Quelength				
Replications: 100	95% C.I.	4.4085	+- 0.4298	Std Dev: 2.1654

Vergleich zu einem Experiment mit 10.000 Läufen

Bei Erfassung der 10.000
Mittelwerte mit einer RV

CI-Bestimmung im Beispiel



ql

qt

et

Start

sample_mean(ql)

sample_mean(qt)

sample_mean(et)

gewünschte Sicherheit

Stichproben-umfang

Makro

Profile, Histogrammme

1-te Stichprobe (Simulationslauf)

...

Profile, Histogrammme

100-te Stichprobe (Simulationslauf)

...

Random_Variable

av_ql

av_qt

av_et

tabulate

report (rv)

Histogrammme

build_mean_ci (...)

Worin Besteht der Unterschied ?

Stop

special_report (...)

Mittelwert

± halbe CI-Breite

Standardabweichung

Confidence Intervals after 100 runs

Elapsed Time

Replications: 100	95% C.I.	3.5150	+- 0.2167	Std Dev: 1.0920

Queueing Time

Replications: 100	95% C.I.	2.2204	+- 0.2107	Std Dev: 1.0619

Average Quelength

Replications: 100	95% C.I.	4.4085	+- 0.4298	Std Dev: 2.1654

Confidence Intervals after 10000 runs

Elapsed Time

Samples: 10000 **95% C.I.:** 3.5793 ... 3.6276 **Std Dev(Mean):** 0.0123

Queueing Time

Samples: 10000 **95% C.I.:** 2.2800 ... 2.3275 **Std Dev(Mean):** 0.0121

Average Quelength

Samples: 10000 **95% C.I.:** 4.5505 ... 4.6489 **Std Dev(Mean):** 0.0251

System Status at Time 0.0000

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrive	9611255	200000	9811255	OFF	0.99
service	9610452	400000	10010452	OFF	1.00

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
rv_mean_elapsed_	10000	3.603	1.234		1.80	15.92
rv_mean_quelengt	10000	4.600	2.511		1.10	30.53
rv_mean_queueing	10000	2.304	1.212		0.60	14.55

Inhalt C.6

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  ODEMx-Elemente
- **C.5**
  Obektorientierung
- **C.6**
  DISCO-Elemente

1. Simula-Bibliothek DISCO
2. SLX-Bibliothek Statistics
   - Modellgrößen, Bewertungsgrößen, Kennwertermittlung
   - Random_Variable, Histogram, Statistics
   - Einfache Simulationsläufe, Histogramm-Auswertung
   - Zeitintervall-basierte Beobachtung und Auswertung
   - Konfidenz-Intervall-Schätzung
   - Sequential Sampling
   - CI-Schätzung mit antithetischen Läufen
   - Vergleich zwischen simulierten Systemvarianten

Methode: Sequential Sampling

- hier wird jeweils **nach einem Simulationslauf** geprüft,
  ob die ermittelten **Konfidenzintervalle** für die Schätzung der Ergebnisgrößen
  hinreichend klein sind

- Vorgehensweise:

  erst nach der Durchführung von **Pilotläufen**, z.B. 10 Läufen,
  werden die Konfidenzintervalle berechnet.

  Es wird dann jeweils ein weiterer Lauf gestartet, wenn das berechnete Konfidenzintervall für diese Anwendung zu groß ist.

Beispiel

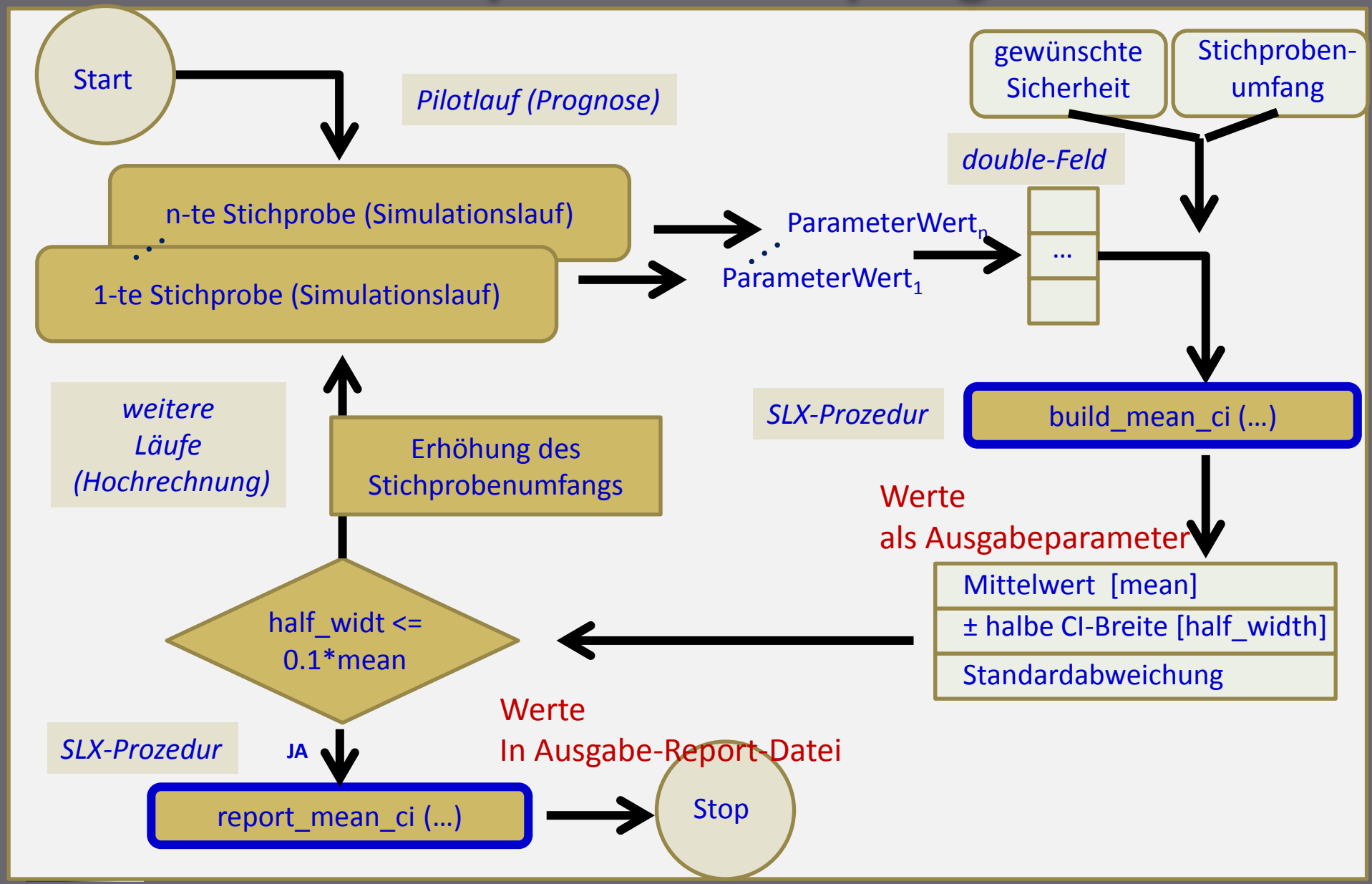
Für das Fallbeispiel Bank fordern wir nun für alle Ergebnisgrößen, dass die halbe Länge des Konfidenzintervalls **nicht größer als 10% des geschätzten Mittelwertes** ist.

Die Größe der Konfidenzintervalle wird mit der SLX-Prozedur

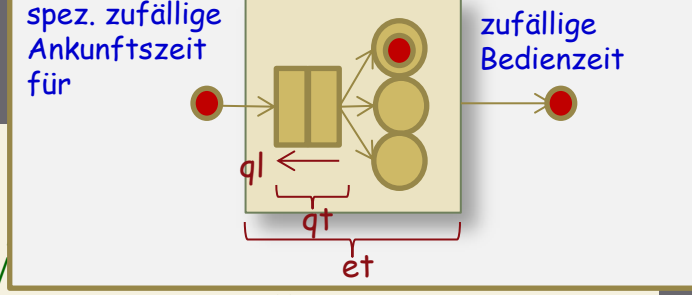
build_mean_ci (...)

berechnet.

Sequential Sampling



Beispiel




```
//*********************************************
// Example EX0067
//*********************************************
import <stats>
import <h7>

module basic {
   rn_stream  arrive,
              service ;
   random_variable
              rv_elapsed_time,
              rv_queueing_time ;
   random_variable (time)
              rv_quelength;

   storage clerk capacity= 3;
   control int in_customer, out_customer;
   int que_length, run ;
   constant float close_time=8*60, service_time=1.3 ;
   constant int max_runs = 10000 ;
   boolean door_closed;

   // for data collecting
   double     sample_elapsed_time [ 1 .. max_runs ],
              sample_queueing_time [ 1.. max_runs ],
              sample_quelength [ 1 .. max_runs ];

   // Break if OK for all confidence intervals
   boolean ok_CI ;
```

```
class cl_customer {
actions {
              in_customer ++;
              que_length++; //
              tabulate rv_quelength= que_length; // tabulate
              enter clerk; // try to catch a clerk
              que_length --; // decrement queue length
              tabulate rv_quelength= que_length count=0 ;
              tabulate rv_queueing_time= time - ACTIVE->mark_time;
              advance rv_expo ( service , service_time ); // service time
              leave clerk;
              out_customer ++;
              tabulate rv_elapsed_time = time - ACTIVE->mark_time;
              }
}
```

```
procedure main()  {
    // 10 pilot runs
    for ( run = 1; run <= 10 ; run ++ ) {
              run_model();      // run the model
              report_model (); // collect statistics for this run (if run >1)
                               // typically no report
              clear_model () ;  // clear the model
     }
    // Run until all confidence interval are OK
    while ( ! ok_CI ) {
              run++;
              run_model();      // run the model
              report_model (); // collect statistics for this run
                               // with report after the last run
              clear_model () ;  // clear the model
    }
} // main
```

```
procedure run_model ()        {
    ...
}
```

```
procedure clear_model() {
                ...
}
```

```
procedure report_model () {
        float mean, deviation , half_width ;
        boolean ok_elapsed_time, ok_queueing_time, ok_quelength ;

        // Collecting data for every run
        sample_elapsed_time [ run ] = sample_mean ( rv_elapsed_time );
        sample_queueing_time [ run ] = sample_mean ( rv_queueing_time );
        sample_quelength [ run ] = sample_mean ( rv_quelength );

        if ( run >1 ) {
                build_mean_ci ( sample_elapsed_time , run , 0.95 , mean , deviation , half_width );
                if (( half_width / mean ) < 0.10)
                        ok_elapsed_time = TRUE;

                build_mean_ci ( sample_queueing_time , run , 0.95 , mean , deviation , half_width );
                if (( half_width / mean ) < 0.10)
                        ok_queueing_time = TRUE;

                build_mean_ci ( sample_quelength , run , 0.95 , mean , deviation , half_width );
                if (( half_width / mean ) < 0.10)
                        ok_quelength = TRUE;

                ok_CI = ( ok_elapsed_time && ok_queueing_time && ok_quelength );
        }

        if ( ok_CI ) {
                report ( system );
                print options= bold , underline (run) "Confidence Intervals after _ runs \n";
                report_mean_ci ( "Elapsed Time ", 0.95, sample_elapsed_time , run );
                report_mean_ci ( "Queueing Time ", 0.95, sample_queueing_time , run );
                report_mean_ci ( "Average Quelength ", 0.95, sample_quelength , run );
        }
}
```

Outp-EX-0067

EX-0067-(Bank, Konfidenzintervall, Mindestintervall).slx: Lines: 7,533 Errors: 0 Warnings: 0 Lines/Second: 343,987 Memory: 5 MB

Execution begins

report(system)
des letzten der 101 Läufe

System Status at Time 480.4873

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrive	95764	200000	295764	OFF	0.20
service	95751	400000	495751	OFF	0.63

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
rv_elapsed_time	979	4.624	4.119		0.02	20.63
rv_quelength	979	6.854	8.157		0.00	35.00
rv_queueing_time	979	3.364	3.857		0.00	14.62

Storage ...

Confidence Intervals after 101 runs

Abbruch-Kriterium:
half_width / mean < 0.1

Elapsed Time
Samples: 101 **95% C.I.:** 3.2548 ... 3.7056 **Std Dev(Mean):** 0.1136

Queueing Time
Samples: 101 **95% C.I.:** 1.9853 ... 2.4116 **Std Dev(Mean):** 0.1074

Average Quelength
Samples: 101 **95% C.I.:** 3.9307 ... 4.7991 **Std Dev(Mean):** 0.2188

Execution complete
Objects created: 45 passive and 95,752 active Pucks created: 95,953 Memory: 5 MB Time: 0.26 seconds

Inhalt C.6

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  ODEMx-Elemente
- **C.5**
  Obektorientierung
- **C.6**
  DISCO-Elemente

1. Simula-Bibliothek DISCO
2. SLX-Bibliothek Statistics
   - Modellgrößen, Bewertungsgrößen, Kennwertermittlung
   - Random_Variable, Histogram, Statistics
   - Einfache Simulationsläufe, Histogramm-Auswertung
   - Zeitintervall-basierte Beobachtung und Auswertung
   - Konfidenz-Intervall-Schätzung
   - Sequential Sampling
   - CI-Schätzung mit antithetischen Läufen
   - Vergleich zwischen simulierten Systemvarianten

Zufallszahlengeneratoren

- **Chi-Quadrat-Test der Gleichverteilung**
  Wahrscheinlichkeit, dass Annahme einer Gleichverteilung für ermittelten Werte z
- Optionales **Histogra** für Verteilung plus
- Einschränkung des **Wertebereiches** vo
- **Antithetische Variation**
  mit x als ZZ ist auch 1–x eine ZZ
  (Erhöhung der Konvergenzgeschw. von Monte-Carlo-Simulationen)

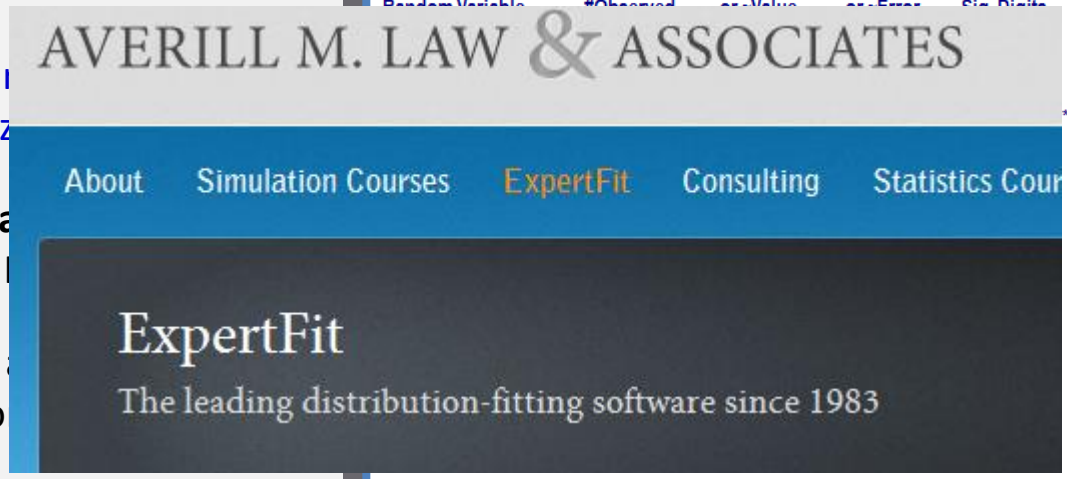
```
rn_stream  rn1 [ seed= …] [antithetic];

random_input arrival=  rv_expo( rn1, 10.0)
          [ accept (lower [, upper]) ]
          histogram start=0 width=6 count=10;
```

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
rn1	14412	200000	214412	OFF	0.91
rn2	4208	400000	404208	OFF	0.82

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
arrival_smallTru	4208	10.274	10.217		0.00	88.85

Lower	Upper	Frequency	Percent	
0.0	6.0	1875	44.558	**********
6.0	12.0	1018	24.192	**********
12.0	18.0	579	13.760	**********
18.0	24.0	337	8.009	********
24.0	30.0	173	4.111	****
30.0	36.0	96	2.281	**
36.0	42.0	53	1.260	*
42.0	48.0	42	0.998	*
48.0	54.0	15	0.356	
54.0	60.0	12	0.285	

Overflow: 8 Average Overflow: 70.64



Methode: Berechnung von Zufallszahlen (Wdh.)

rn_stream arrive **seed**= 123456;

frn (arrive); //liefert bei jedem Aufruf
// neue [0,1-Zufallszahl

- Theoretische Verteilungsfunktionen
- Empirische Verteilungsfunktionen
- Bezier-Verteilungsfunktionen

- Beschneidung der Randbereiche

Erzeugung positiver gleichverteilter int-Werte [0, maxInt)

Transformation in gleichverteilte Gleitkommazahlen (double) [0.0, 1,0)

spezifische Transformation in die gewünschte Zielverteilungs-funktion

evtl. Adaption

41 Verteilungsfunktionen

Beispiel:
rv_expo (arrive); //liefert bei jedem Aufruf
// neue Zufallszahl entspr. Exponentialverteilung

Antithetische SLX-Methode zur CI-Bestimmung

aus der Klasse der Varianz-Reduktionstechniken

1. Durchführung von n normalen (stochastisch unabhängigen) Läufen
   (**clear** setzt die ZZ-Startwerte nicht zurück),
   Die zu bestimmenden Parameter werden für jeden Lauf gespeichert

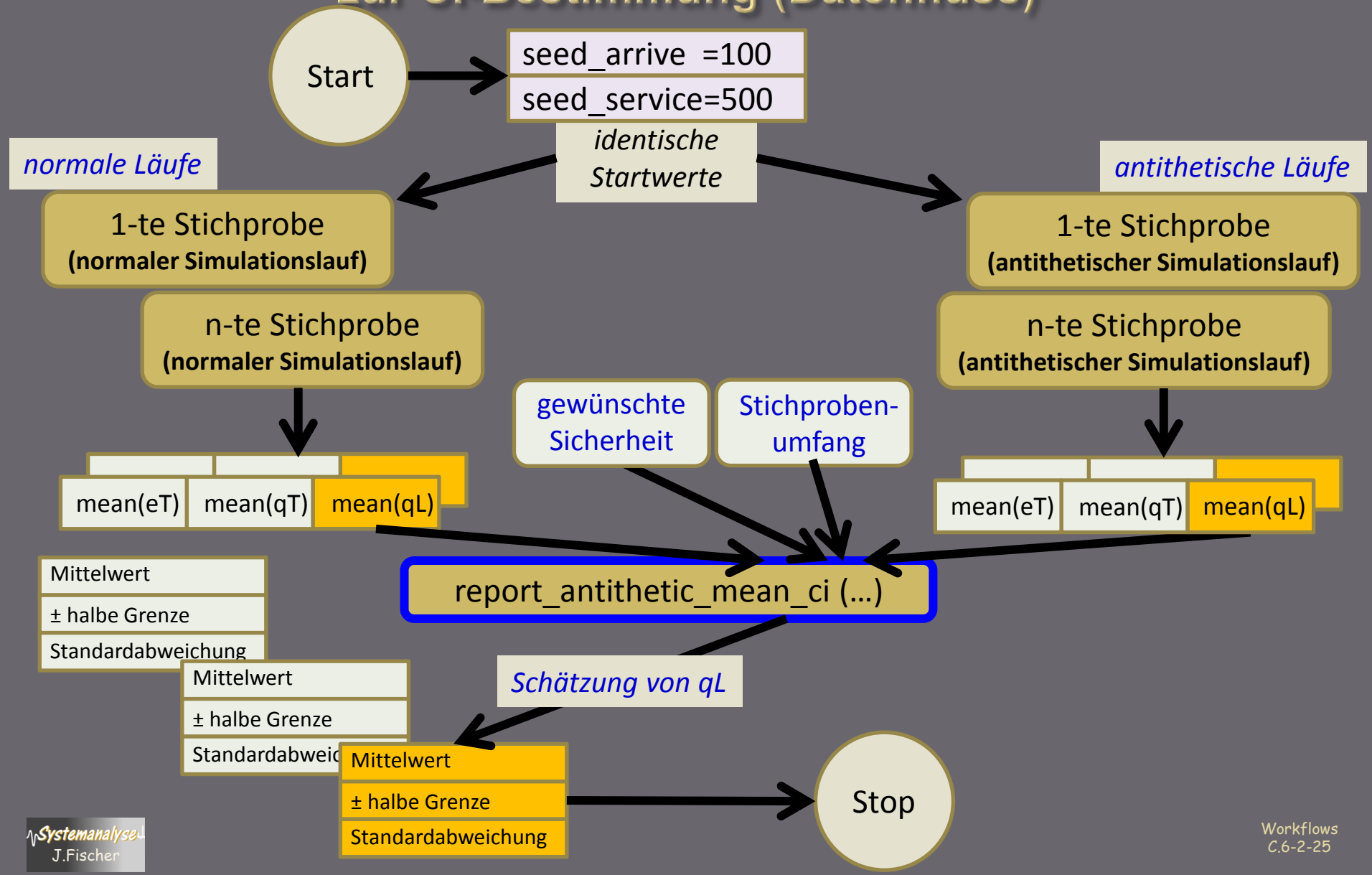
2. Durchführung von n antithetischen (stochastisch unabhängigen) Läufen, d.h. die Zufallszahlen müssen antithetisch sein zu denen der normalen Läufe:
   $x \leftrightarrow (1-x)$

3. Berechnung der Mittelwerte je Parameter aus normalen und antithetischen Läufen

   Da eine hohe negative Korrelation zwischen den beiden Mittelwerten besteht, wird eine hohe Varianzreduktion erreicht,
   infolge dessen sich das **Konfidenzintervall** verkleinert

4. Auf Basis dieser Mittelwerte erfolgt die Schätzung der Konfidenzintervalle

Antithetische SLX-Methode
zur CI-Bestimmung (Datenfluss)



Umsetzung in SLX

- Antithetische Zufallszahlen lassen sich in SLX erzeugen
- Unabhängigkeit wird durch Startwerte der ZG garantiert
- SLX-Prozeduren für

1. Berechnung des Konfidenzintervalls und Schätzung des Mittelwertes unter Nutzung normaler und antithetischer Läufe

```
build_antithetic_mean_ci (    samples1[*], samples2[*],
                              scount, level,
                              smean, stdev, half_width   )
```

2. Berechnung und Ausgabe des Konfidenzintervalls

```
report_antithetic_mean_ci (   titl,
                              level,
                              samples1[*], samples2[*],
                              scount                     )
```

```
//*********************************************
// Example EX0025
//*********************************************
import <stats>
import <h7>

module basic {
    rn_stream   arrive, service ;
    random_variable rv_elapsed_time,
                    rv_queueing_time ;
    random_variable (time) rv_quelength;

    storage clerk capacity=3;

    // runs with normal and antithetic random_streams
    type t_run_type enum {normal , antithetic};
    t_run_type run_type;

    control int in_customer, out_customer;
    int que_length, run ;
            constant float   close_time=8*60,
                             service_time=1.3 ;
            constant int     n_runs = 20 ;
            boolean          door_closed;

            // sampling data from every run
            float            sample_elapsed_time [ 1 .. n_runs ],
                             sample_queueing_time [ 1 .. n_runs ],
                             sample_quelength [ 1 .. n_runs ],
                             anti_sample_elapsed_time [ 1 .. n_runs ],
                             anti_sample_queueing_time [ 1 .. n_runs ],
                             anti_sample_quelength [ 1 .. n_runs ];

            // first seeds for first run
            int              seed_arrive = 100,
                             seed_service = 500;
```

```
class cl_customer {
actions {
                ...
}
```

```
procedure main()  {
    for ( run_type = each t_run_type ) {
            for ( run = 1; run <= n_runs ; run ++ ) {
                rn_stream_setting () ;      // preparing the random_streams
                run_model();                // run the model
                report_model ();            // collect statistics for this run
                clear_model () ;            // clear the model
            }
    }
    final_report (); // calculating CI based on normal and antithetic runs
} // main
```

Synchronität der Startwerte

```
procedure rn_stream_setting () {
        if ( run_type == normal ) { // seed for normal runs
                rn_seed arrive= (seed_arrive + run*100000);
                rn_seed service = (seed_service + run*100000);

        }
        else  { // seed for antithetic runs
                rn_seed arrive= (seed_arrive + run*100000) antithetic;
                rn_seed service = (seed_service + run*100000) antithetic;
        }
}
```

```
procedure run_model ()          {
    ...
}
```

```
procedure clear_model ()        {
    ...
}
```

```
procedure report_model () {
        print ( run ) "Run _ finished \n";
         // Collecting data for every run
         if ( run_type == normal ) { // normal randoms
                    sample_elapsed_time [ run ] = sample_mean ( rv_elapsed_time );
                    sample_queueing_time [ run ] = sample_mean ( rv_queueing_time );
                    sample_quelength [ run ] = sample_mean ( rv_quelength );
          }
          else  { // antithetic randoms
                    anti_sample_elapsed_time [ run ] = sample_mean ( rv_elapsed_time );
                    anti_sample_queueing_time [ run ] = sample_mean ( rv_queueing_time );
                    anti_sample_quelength [ run ] = sample_mean ( rv_quelength );
          }
          if ( run == n_runs )
                    report system;
}
```

n_runs== 20

```
procedure final_report () {
        print (n_runs) "Confidenc Intervals after _ antithetic runs \n";

        report_antithetic_mean_ci ( "Elapsed Time ", 0.95,
                                    sample_elapsed_time, anti_sample_elapsed_time ,n_runs );
        report_antithetic_mean_ci ( "Queueing Time ", 0.95,
                                     sample_queueing_time , anti_sample_queueing_time, n_runs );
         report_antithetic_mean_ci ( "Average Quelength ", 0.95,
                                     sample_quelength , anti_sample_quelength , n_runs );
}
```

EX-0025-(Bank, Varianzreduktion).slx: Student SLX-32 OV031 Lines: 7,555 Errors: 0 Warnings: 0 Lines/Second: 341,205 Memory: 3 MB

Execution begins

Run 1 finished
...
Run 20 finished

System Status at Time 482.2518

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrive	934	2000100	2001034	OFF	0.36
service	934	2000500	2001434	OFF	0.49

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
rv_elapsed_time	934	3.539	3.267		0.01	16.76
rv_quelength	1868	4.273	5.945		0.00	27.00
rv_queueing_time	934	2.206	2.957		0.00	13.63

Storage	Capacity	%Util	%Util	%Util	Entries	Time/Puck	Status	%Avail	Contents
clerk	3	86.06	934	1.33	AVAIL	100.00	2.58	0	3

Run 1 finished
...
Run 20 finished

System Status at Time 481.2090

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrive	974	2000100	2001074	ON	0.34
service	974	2000500	2001474	ON	0.52

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
rv_elapsed_time	974	2.852	2.318	0.00	14.21	
rv_quelength	1948	3.234	4.101	0.00	18.00	
rv_queueing_time	974	1.598	1.950	0.00	8.95	

Storage	Capacity	%Util	%Util	%Util	Entries	Time/Puck	Status	%Avail	Contents
clerk	3	84.61	974	1.25	AVAIL	100.00	2.54	0	3

Confidenc Intervals after 50 antithetic runs

Elapsed Time
Antithetic Samples: 50 **95% C.I.** 3.5309 ... 3.9986 **Std Dev(Mean):** 0.1163
Correlation between sample means: -0.27

Queueing Time
Antithetic Samples: 50 **95% C.I.** 2.2309 ... 2.6971 **Std Dev(Mean):** 0.1160
Correlation between sample means: -0.26

Average Quelength
Antithetic Samples: 50 **95% C.I.** 4.4353 ... 5.3475 **Std Dev(Mean):** 0.2269
Correlation between sample means: -0.27
Execution complete
Objects created: 45 passive and 96,244 active Pucks created: 96,445 Memory: 5 MB Time: 0.51 seconds

Confidenc Intervals after 100 antithetic runs

Elapsed Time
Antithetic Samples: 100 **95% C.I.** 3.6354 ... 3.9240 **Std Dev(Mean):** 0.0727
Correlation between sample means: -0.19

Queueing Time
Antithetic Samples: 100 **95% C.I.** 2.3330 ... 2.6198 **Std Dev(Mean):** 0.0722
Correlation between sample means: -0.18

Average Quelength
Antithetic Samples: 100 **95% C.I.** 4.6242 ... 5.1960 **Std Dev(Mean):** 0.1441
Correlation between sample means: -0.20
Execution complete
Objects created: 45 passive and 192,154 active Pucks created: 192,555 Memory: 5 MB Time: 0.68 seconds

Confidenc Intervals after 20 antithetic runs

Elapsed Time

Antithetic Samples: 20 **95% C.I.** 3.4381 ... 4.0409 **Std Dev(Mean):** 0.1440
Correlation between sample means: -0.23

Elapsed Time
Samples: 101 **95% C.I.:** 3.2548 ... 3.7056 **Std Dev(Mean):** 0.1136

Queueing Time

Antithetic Samples: 20 **95% C.I.** 2.1375 ... 2.7330 **Std Dev(Mean):** 0.1422
Correlation between sample means: -0.21

Queueing Time
Samples: 101 **95% C.I.:** 1.9853 ... 2.4116 **Std Dev(Mean):** 0.1074

Average Quelength

Antithetic Samples: 20 **95% C.I.** 4.1993 ... 5.4148 **Std Dev(Mean):** 0.2903
Correlation between sample means: -0.21

Average Quelength
Samples: 101 **95% C.I.:** 3.9307 ... 4.7991 **Std Dev(Mean):** 0.2188

Execution complete
Objects created: 45 passive and 38,227 active Pucks created: 38,308 Memory: 3 MB Time: 0.48 seconds

Inhalt C.6

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  ODEMx-Elemente
- **C.5**
  Obektorientierung
- **C.6**
  DISCO-Elemente

1. Simula-Bibliothek DISCO
2. SLX-Bibliothek Statistics
   - Modellgrößen, Bewertungsgrößen, Kennwertermittlung
   - Random_Variable, Histogram, Statistics
   - Einfache Simulationsläufe, Histogramm-Auswertung
   - Zeitintervall-basierte Beobachtung und Auswertung
   - Konfidenz-Intervall-Schätzung
   - Sequential Sampling
   - CI-Schätzung mit antithetischen Läufen
   - Vergleich zwischen simulierten Systemvarianten

SLX-Methode zum Systemvariantenvergleich

Ziel: Vergleich von zwei Systemvarianten.

1. für jede Variante werden n Simulationsläufe durchgeführt und
2. Differenzbildung der Werte für jeden Lauf

mit Bestimmung eines Konfidenzintervalls für die Differenz

Voraussetzung:

synchronisierte Ströme von Zufallszahlen für beide Varianten
(es sollen nur strukturbedingte Unterschiede erfasst werden,
und nicht zufallsbedingte)

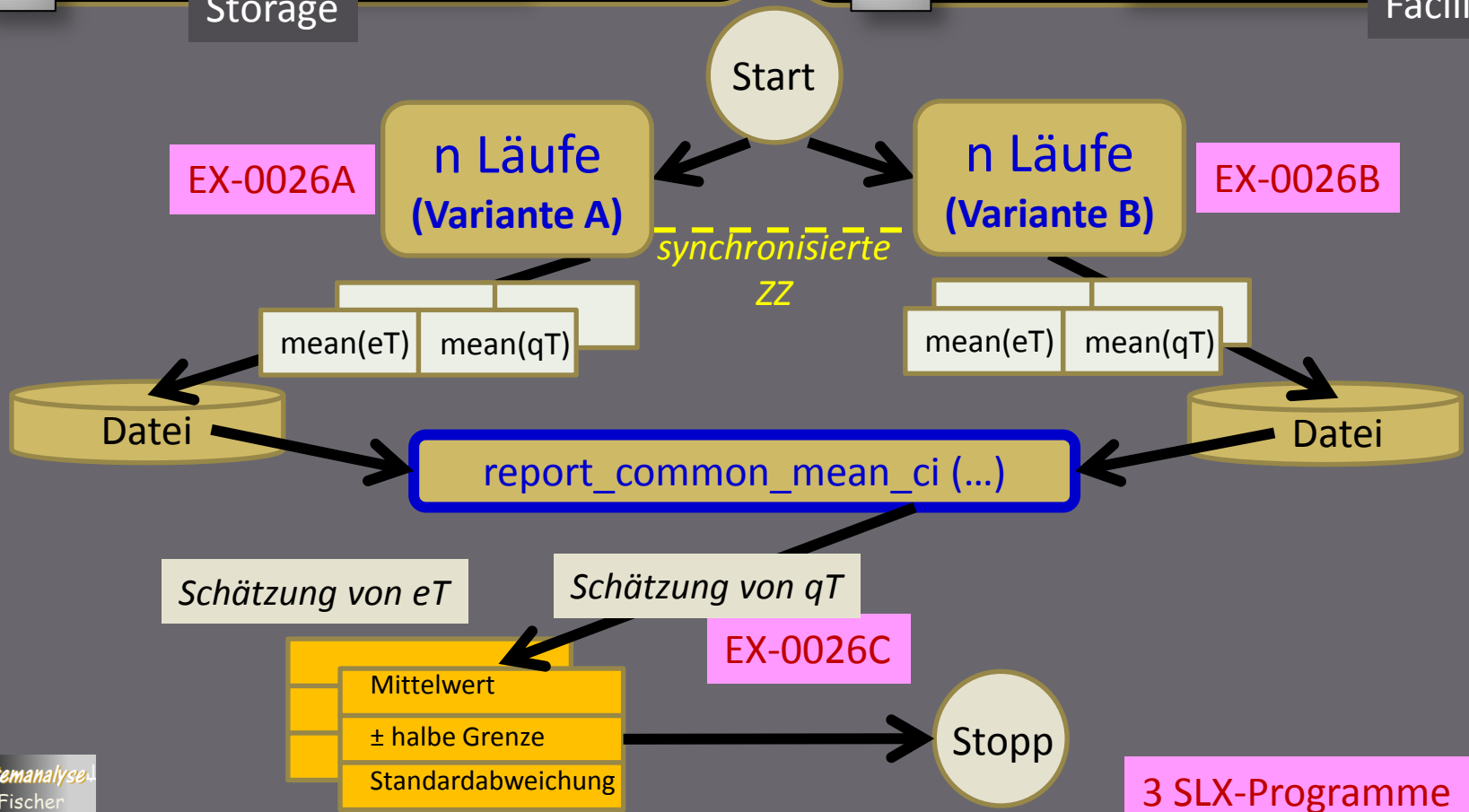
Beispiel: Bankschalter A und B

A) gemeinsame WS für 3 Angestellte

B) jeder Angestellte hat eine individuelle WS
Kunde wählt bei Ankunft die kürzeste ohne später wechseln zu können

Bewertungsgrößen: mittl. Wartezeit, mittl. Gesamtverweilzeit

Datenfluss bei der Berechnung



3 SLX-Programme

Umsetzung in SLX

- SLX-Prozeduren für

1. Berechnung des Konfidenzintervalls und Schätzung des Mittelwertes der Differenz zweier Varianten, sowie des Korrelationskoeffizienten

```
build_common_mean_ci (  samples1[*], samples2[*],
                        scount, level,
                        smean, stdev, half_width ,
                        corr )
```

2. Berechnung und Ausgabe des Konfidenzintervalls

```
report_common_mean_ci ( titl,
                        level,
                        samples1[*], samples2[*],
                        scount                    )
```

```
//*****************************************************************************
// Example EX0026C
//*****************************************************************************
import <stats>
import <h7>
module basic {
        filedef  resultA name="EX0026A.DAT";
        filedef  resultB name="EX0026B.DAT";

        float    sample_A_elapsed_time [1..200],
                 sample_A_queueing_time [1..200],
                 sample_B_elapsed_time [1..200],
                 sample_B_queueing_time [1..200];
        int      count_A, count_B, i ;

        procedure main() {
                // read values for sample case A
                i = 1;
                forever {
                        read file=resultA end=finish_A
                                ( sample_A_elapsed_time [i],
                                  sample_A_queueing_time [i]);
                        i++;
                }
                finish_A : count_A = --i;
                i = 1;

                // read values for sample case B
                forever {
                        read file=resultB end=finish_B
                                ( sample_B_elapsed_time [i],
                                  sample_B_queueing_time [i]);
                        i++;
                }
                finish_B : count_B = --i;

                // The same sample counter
                if ( count_A != count_B ) {
                        print ( count_A , count_B )
                            "sample size A : ***** "
                            "sample size B : ****** \n";
                        exit ( 0) ;
                }
                report_common_mean_ci (
                    "Difference between Case A and Case B for elapsed time ",
                     0.95 ,
                     sample_A_elapsed_time ,
                     sample_B_elapsed_time ,
                     count_A );
                report_common_mean_ci (
                    "Difference between Case A and Case B for queueing time ",
                    0.95 ,
                    sample_A_queueing_time ,
                    sample_B_queueing_time ,
                    count_A );
        } // main
}
```

EX-0026C.slx: SLX-64 OV031 Lines: 7,471 Errors: 0 Warnings: 0 Lines/Second: 267,837 Memory: 4 MB
File C:\Users\fischer\Documents\Lehre\Workflow-Projekt\SLX-Programme\EX-0026C.slx saved
Execution begins

Difference between Case A and Case B for elapsed time

Common Samples: 100 **95% C.I.** -0.6407 ... -0.5977 **Std Dev(Mean):** 0.0108
Correlation between sample means: 0.99

Elapsed Time
CI ist < 0.0 ➔ elapsed_time (B) > elapsed_time (A)

Difference between Case A and Case B for queueing time

Common Samples: 100 **95% C.I.** -0.6407 ... -0.5977 **Std Dev(Mean):** 0.0108
Correlation between sample means: 0.99

QueuingTime
CI ist < 0.0 ➔ queuing_time (B) > queuing_time(A)

Execution complete
Objects created: 23 passive and 1 active Pucks created: 2 Memory: 4 MB Time: 0.07 seconds

Die mittleren Wartezeiten und die mittleren Verweilzeiten sind in der Variante B (Vorabauswahl) größer als in der Variante A.

Grund: Das Konfidenzintervall für den Mittelwert der Differenz ist kleiner Null. Somit sind die Ergebnisgrößen für Variante B größer als die für Variante A.

```
//*************************************************************
// Example EX0026A
//*************************************************************
import <stats>
import <h7>
module basic {
        rn_stream arrive, service ;
        random_variable rv_elapsed_time,
        rv_queueing_time ;

        storage clerk capacity=3;

        control int in_customer, out_customer;
        int run ;
        constant float close_time=8*60, service_time=1.3 ;
        constant int n_runs = 100 ;
        boolean door_closed;

        filedef result name="EX0026A.DAT";

        // first seeds for first run
        int seed_arrive = 100,
        seed_service = 500;

        class cl_customer {
                float serv_time;
                actions{
                        in_customer ++; // increment customer counter
                        serv_time = rv_expo ( service , service_time );
                        enter clerk; // try to catch a clerk
                        tabulate rv_queueing_time= time - ACTIVE->mark_time;
                        advance serv_time; // service time
                        leave clerk;
                        out_customer ++;
                        tabulate rv_elapsed_time = time - ACTIVE->mark_time;
                }
        }
```

```
procedure main() {
        for ( run = 1; run <= n_runs ; run ++ ) {
                rn_stream_setting () ; // preparing random streams
                run_model(); // run the model
                report_model (); // collect statistics for this run
                clear_model () ; // clear the model
        }
}// main

procedure rn_stream_setting () {
        rn_seed arrive= (seed_arrive + run*100000);
        rn_seed service = (seed_service + run*100000);
}

procedure run_model () {
        float intensity=2.0;
        fork { // Arriving Customer
                forever {
                        advance rv_expo ( arrive , 1/intensity );
                        activate new cl_customer;
                        if ( door_closed ) terminate;
                }
        }
        fork  {  // Controlling the bank
                advance close_time;
                door_closed = TRUE;
                terminate;
        }
}
```

```
procedure run_model () {
     float intensity=2.0;
     fork { // Arriving Customer
                    forever {
                                   advance rv_expo ( arrive , 1/intensity );
                                   activate new cl_customer;
                                   if ( door_closed ) terminate;
                    }
     }
     fork  {  // Controlling the bank
                    advance close_time;
                    door_closed = TRUE;
                    terminate;
     }
     wait until ( (time > close_time) && ( in_customer == out_customer ));
}

procedure clear_model() {
     clear system; // Clearing for SLX - features
     /* Clearing model specific variables */
    in_customer = 0;
    out_customer = 0;
    door_closed = FALSE ;
}

procedure report_model () {
      print ( run ) "Run _ finished \n";
     // write data for every run
     write file=result ( sample_mean ( rv_elapsed_time ),
                    sample_mean ( rv_queueing_time ) )
                    "_._____ _._________ \n" ;
     if ( run ==n_runs ) {
                    print " Report for Case A \n";
                    report ( system );
   }
}
```

```
//*********************************************
// Example EX0026B
//*********************************************
import <stats>
import <h7>
module basic {
        rn_stream arrive, service ;
        random_variable rv_elapsed_time,
        rv_queueing_time ;

        facility clerk [3];
        control int in_customer,
                    out_customer;
        int     run ,
                que_length[ 3 ] ,
                min,
                i ;

        constant float      close_time=8*60,
                            service_time=1.3 ;
        constant int n_runs = 100 ;
        boolean door_closed;

        filedef result name="EX0026A.DAT";

        // first seeds for first run
        int seed_arrive = 100,
        seed_service = 500;
```

```
class cl_customer {
      int my_number;
      float serv_time;
      actions {
                  in_customer ++; // increment customer counter
                  serv_time = rv_expo ( service , service_time );
                  my_number = 1;
                  for ( i=2; i<=3 ; i++ ) {
                        if ( que_length[i] < que_length[ my_number ] )
                        my_number = i;
                  }
                  que_length[ my_number ] ++ ; // increment queue length
                  seize clerk[ my_number ]; //wait for service
                  que_length[ my_number ] -- ; // decrement queue length
                  tabulate rv_queueing_time= time - ACTIVE->mark_time;
                  advance serv_time; // service time
                  release clerk [ my_number ];
                  out_customer ++;
                  tabulate rv_elapsed_time = time - ACTIVE->mark_time;
      }
}
```

```
procedure main () {
      ...
}

procedure run_model () {
      ...
}

procedure clear_model() {
      ...
}

procedure report_model () {
      ...
}
```