# SGI RandomSample*

```
template <class _ForwardIter, class _OutputIter, class _Distance>
_OutputIter random_sample_n(_ForwardIter __first, _ForwardIter __last,
                            _OutputIter __out, const _Distance __n)

template <class _ForwardIter, class _OutputIter, class _Distance, class _RNG>
_OutputIter random_sample_n(_ForwardIter __first, _ForwardIter __last,
                            _OutputIter __out, const _Distance __n,
                            _RandomNumberGenerator& __rand)

template <class _InputIter, class _RandomAccessIter>
inline _RandomAccessIter
random_sample(_InputIter __first, _InputIter __last,
              _RandomAccessIter __out_first, _RandomAccessIter __out_last)

template <class _InputIter, class _RandomAccessIter, class _RNG>
inline _RandomAccessIter
random_sample(_InputIter __first, _InputIter __last,
              _RandomAccessIter __out_first, _RandomAccessIter __out_last,
              _RandomNumberGenerator& __rand)
```

# SGI RandomSample*

```
template <class _InputIter, class _RandomAccessIter, class _Distance>
_RandomAccessIter __random_sample(_InputIter __first, _InputIter __last,
                                  _RandomAccessIter __out,
                                  const _Distance __n)
{
  _Distance __m = 0;
  _Distance __t = __n;
  for ( ; __first != __last && __m < __n; ++__m, ++__first)
    __out[__m] = *__first;

  while (__first != __last) {
    ++__t;
    _Distance __M = __random_number(__t);
    if (__M < __n)
      __out[__M] = *__first;
    ++__first;
  }

  return __out + __m;
}
```

# Initial Attempt

```
template <class InputIter, class RandomAccessIter, class RNG, class Distance>
RandomAccessIter __random_sample(InputIter first, InputIter last, RandomAccessIter out,
    RNG& rand, Distance n)
{
   BOOST_CONCEPT_ASSERT(( boost::InputIterator<InputIter> ));
   BOOST_CONCEPT_ASSERT(( boost::Mutable_RandomAccessIterator<RandomAccessIter> ));
   BOOST_CONCEPT_ASSERT(( boost::UnaryFunction<RNG, Distance, Distance> ));

   Distance m(0);
   Distance t(n);
   for ( ; first != last && m < n; ++m, ++first)
      out[m] = *first;

   while (first != last) {
      ++t;
      const Distance M = rand(t);
      if (M < n)
         out[M] = *first;
      ++first;
   }

   return out + m;
}
```

# Considerations

- License
  - HP (1994) and SGI (1996)
- Sufficient entropy?
- Random number generator
  - Default to rand?  drand48?  mt1337?
  - see e.g., std::random_shuffle, rand() in vc10

# Goals

- Determine if in-kind port is “good enough”
  - Improve entropy?
- Remove RA output iterator precondition
  - Enable forward iterators, iterator adaptors
- Address RNG question